目　　录

第1章 概 述

1.1 图像处理简介

人类传递信息的主要媒介是语音和图像。据统计，在人类接收的信息中，听觉信息占20%，视觉信息占60%，其他如味觉、触觉、嗅觉总的加起来不过占20%。因此，作为传递信息的重要媒介和手段——图像信息是十分重要的，俗话说“百闻不如一见”、“一目了然”，都反映了图像在传递信息中的独到之处。视觉是人类最高级的感知器官，所以，毫无疑问图像在人类感知中扮演着最重要的角色。然而，人类感知只限于电磁波谱的视觉波段，成像机器则可覆盖几乎全部电磁波谱，从伽马射线到无线电波。它们可以对非人类习惯的那些图像源进行加工，这些图像源包括超声波、电子显微镜及计算机产生的图像。因此，数字图像处理涉及各种各样的应用领域。

图像处理最早的应用之一是在报纸业，当时，图像第一次通过海底电缆从伦敦传往纽约。早在20世纪20年代曾引入Bartlane电缆图片传输系统，把横跨大西洋传送一幅图片所需的时间从一个多星期减少到3h。为了用电缆传输图片，首先进行了压缩编码，然后在接收端用特殊的打印设备重构该图片。

大规模的存储和显示系统以及数据处理技术的发展，为数字图像处理的实现奠定了客观的物理实现基础。第一台可以执行有意义的图像处理任务的大型计算机出现在20世纪60年代早期。数字图像处理技术的诞生可追溯至这一时期这些机器的使用和空间项目的开发，这两大发展把人们的注意力集中到数字图像处理的潜能上。利用计算机技术改善空间探测器发回的图像的工作，始于1964年美国加利福尼亚的喷气推进实验室。当时由“旅行者7号”卫星传送的月球图像由一台计算机进行了处理，以校正航天器上电视摄像机中各种类型的图像畸变。图1-1显示了由“旅行者7号”于1964年7月31日上午（美国东部白天时间）9点09分在光线影响月球表面前约17min时摄取的第一张月球图像（痕迹，称为网状痕迹，用于几何校正），这也是美国航天器取得的第一幅月球图像。“旅行者7号”传送的图像可作为改善的增强和复原图像（例如来自“探索者”登月飞行、“水手号”系列空间探测器以及阿波罗载人登月飞行的图像）的基础。

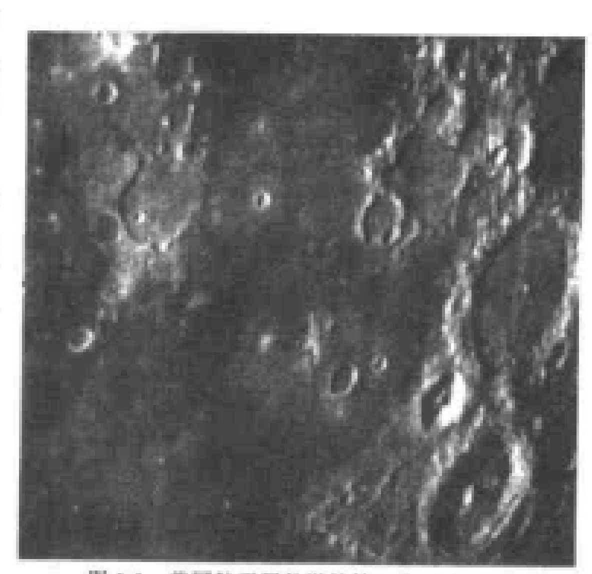


图1-1 美国航天器传送的第一张月球图片

其后，数字图像处理技术发展迅速，目前已成为工程学、计算机科学、信息科学、统计学、物理学、化学、生物学、医学甚至社会科学等领域学习和研究的对象。如今图像处理技术已给人

类带来了巨大的经济和社会效益。不久的将来它不仅在理论上会有更深入的发展，还将成为科学研究、社会生产乃至人类生活中不可缺少的强有力工具。

图像处理科学对人类具有重要意义，它表现在如下 3 个方面：

（1）图像是人们从客观世界获取信息的重要来源

人类是通过感觉器官从客观世界获取信息的，即通过耳、目、口、鼻、手，通过听、看、味、嗅和触摸的方式获取信息。在这些信息中，视觉信息占 60%。视觉信息的特点是信息量大，传播速度快，作用距离远，有心理和生理作用，加上大脑的思维和联想，具有很强的判断能力。其次是人的视觉十分完善，人眼灵敏度高，鉴别能力强，不仅可以辨别景物，还能辨别人的情绪，由此可见，图像信息对人类来说是十分重要的。

（2）图像信息处理是人类视觉延续的重要手段

众所周知，人的眼睛只能看到可见光部分，但就目前科技水平看，能够成像的并不仅仅是可见光。一般来说可见光的波长为 0.38～0.8μm，而迄今为止人类发现可成像的射线已有多种，如（附波长范围）

Y 射线：0.003～0.03nm；

X 射线：0.03～3nm；

紫外线：3～300nm；

红外线：0.8～300μm；

微波：0.3～100cm。

这些射线均可以成像。利用图像处理技术把这些不可见射线所成图像加以处理并转换成可见图像，实际上大大延伸了人类视觉器官的功能，提高了人类认识客观世界的能力。

（3）图像处理技术对国计民生有重要意义

图像处理技术发展到今天，许多技术已日趋成熟。在各个领域的应用取得了巨大的成功和显著的经济效益。如在工程领域、工业生产、军事、医学以及科学研究中的应用已十分普遍。通过分析资源卫星得到的照片可以获得地下矿藏资源的分布及埋藏量；利用红外线、微波遥感技术可侦察到隐蔽的军事设施；X 射线 CT 已广泛应用于临床诊断，由于它可得到人体内部器官的断层图像，因此可准确地确定病灶位置，为诊断和治疗疾病带来了极大的方便。至于在工业生产中的设计自动化及产品质量检验中更是大有可为。在安全保障及监控方面图像处理技术更是不可缺少的基本技术，类似的应用例子随处可见。至于在通信及多媒体技术中图像处理更是重要的关键技术。因此，图像处理技术在国计民生中的重要意义是显而易见的。正因为如此，图像处理理论和技术受到了各界的广泛重视，科学工作者经过不懈的努力，已取得了令人瞩目的成就，并正在向更加深入及更高的层次发展。

1.2　图像处理技术

1.2.1　图像处理技术的分类

一幅图像可定义为一个二维函数 $f(x, y)$，这里 x 和 y 是空间坐标，而在任何一对空间坐标 (x, y) 上的幅值 f 称为该点图像的强度或灰度。当 x、y 和幅值 f 为有限的、离散的数值时，

称该图像为数字图像。数字图像处理是指借用数字计算机处理数字图像，值得提及的是数字图像是由有限的元素组成的，每一个元素都有一个特定的位置和幅值，这些元素称为图像元素、画面元素或像素。像素是广泛用于表示数字图像元素的词汇。而另一类更早期发展的图像类型为模拟图像。

为此，图像处理技术一般也分为两大类：模拟图像处理和数字图像处理。

- 模拟图像处理

模拟图像处理（Analog Image Processing）包括光学处理（利用透镜）和电子处理，如照相、遥感图像处理、电视信号处理等。模拟图像处理的特点是速度快，一般为实时处理，理论上讲可达到光速，并可同时并行处理。电视图像是模拟信号处理的典型例子，它处理的是25 帧/秒的活动图像。模拟图像处理的缺点是精度较差，灵活性差，很难有判断能力和非线性处理能力。

- 数字图像处理

数字图像处理（Digital Image Processing）一般都用计算机处理或实时的硬件处理，因此也称之为计算机图像处理（Computer Image Processing）。其优点是处理精度高，处理内容丰富，可进行复杂的非线性处理，有灵活的变通能力，一般来说只要改变软件就可以改变处理内容。其缺点是处理速度还是一个问题，特别是进行复杂的处理更是如此。一般情况下处理静止画面居多，如果实时处理一般精度的数字图像需要具有 100MIPS 的处理能力；其次是分辨率及精度尚有一定限制，如一般精度图像是 512×512×8bit，分辨率高的可达 2048×2048×12bit，如果精度及分辨率再高，所需处理时间将显著地增加。

广义上讲，一般的数字图像很难为人所理解，因此，数字图像处理也离不开模拟技术，为实现人—机对话和自然的人机接口，特别是需要人去参与观察和判断的情况下，模拟图像处理技术是必不可少的。

1.2.2　数字图像处理的特点

数字图像处理的特点表现在如下几个方面。

（1）图像信息量大

在数字图像处理中，一幅图像可看成是由图像矩阵中的像素（pixel）组成的，每个像素的灰度级至少要用 6bit（单色图像）来表示，一般采用 8bit（彩色图像），高精度的可用 12bit 或 16bit。一般分辨率的图像像素数为 256×256、512×512，高分辨率图像像素数可达 1024×1024 或 2048×2048。例如：

256×256×8bit≈64kB

512×512×8bit≈256kB

1024×1024×8bit≈1MB

2048×2048×8bit≈4MB

X 射线照片一般用 64kB～256kB 的数据量，一幅遥感图像 3240×2340×4B≈30MB，因此，大数据量给存储、传输和处理都带来巨大的困难。

（2）图像处理技术综合性强

在数字图像处理中涉及的基础知识和专业技术相当广泛。一般来说涉及通信技术、计算机技术、电子技术、电视技术，至于涉及的数学、物理学等方面的基础知识就更多。

当今的图像处理理论大多是通信理论的推广，只是把通信中的一维问题推广到二维，以便于分析，在此基础上，逐步发展自己的理论体系。因此，图像处理技术与通信技术休戚相关。

在图像处理工程中的信息获取和显示技术主要源于电视技术，其中的摄像、显示、同步等各项技术是必不可少的。

计算机已是图像处理的常规工具，在图像处理中涉及到软件、硬件、网络、接口等多项技术，特别是并行处理技术在实时图像处理中显得十分重要。

图像处理技术的发展涉及越来越多的基础理论知识，雄厚的数理基础及相关的边缘学科知识对图像处理科学的发展有越来越大的影响。总之，图像处理科学是一项涉及多学科的综合性科学。

（3）图像信息理论与通信理论密切相关

早在 1948 年，Shannon 发表了“A Mathematical Theory of Communication”（通信中的数学理论）一文，奠定了信息论的基础。此后，信息理论渗透到了各个领域。图像信息论也属于信息论科学中的一个分支。从当今的理论发展看，我们可以说图像信息论是在通信理论研究的基础上发展起来的。图像理论是把通信中的一维问题推广到二维空间上来研究的，也就是说，通信研究的是一维时间信息，图像研究的是二维空间信息；通信研究的是时间域和频率域的问题，图像理论研究的是空间域和空间频率域（或变换域）之间的关系；通信理论中认为任何一个随时间变化的波形都是由许多频率不同、振幅不同的正弦波组合而成，图像理论认为任何一幅平面图像是由许多频率、振幅不同的 x–y 方向的空间频率波相叠加而成，高空间频率波决定图像的细节，低空间频率波决定图像的背景和动态范围。

总之，通信中的一维问题都可推广到二维，尽管有些理论尚不完全贴切，但对图像自身理论体系的形成有极大的借鉴意义。

1.2.3 数字图像处理的主要内容

完整的数字图像处理工程大体上可分为如下几个方面：图像信息的获取，图像信息的存储，图像信息的传送，图像信息处理，图像信息的输出和显示。其中，本书讲述的是 MATLAB7.0 在图像信息处理中的应用，主要内容包括 MATLAB 中图像的读写、显示等基本操作和图像处理相关操作。

数字图像处理概括地说主要包括如下几项内容：几何处理（Geometrical Processing），算术处理（Arithmetic Processing），图像增强（Image Enhancement），图像复原（Image Restoration），图像重建（Image Reconstruction），图像编码（Image Encoding），模式识别（Image Recognition），图像理解（Image Understanding）。至于图像变换、图像邻域操作等则是大多数处理操作中频繁应用的基本步骤。

（1）几何处理

几何处理主要包括坐标变换，图像的放大、缩小、旋转、移动，多个图像配准，全景畸变校正，扭曲校正，周长、面积、体积计算等。

（2）算术处理

算术处理主要对图像施以加、减、乘、除等运算，如医学图像的减影处理就有显著的效果。

（3）图像增强

图像增强处理主要是突出图像中感兴趣的信息，而减弱或去除不需要的信息，从而使有用信息得到加强，便于区分或解释。主要方法有直方图增强、伪彩色增强法（PseudoColor）、灰度窗口等技术。

（4）图像复原

图像复原处理的主要目的是去除干扰、模糊和图像畸变，恢复图像的本来面目。典型的去噪操作就属于复原处理。图像噪声包括随机噪声和相干噪声，随机噪声干扰表现为麻点干扰，相干噪声表现为网纹干扰。去模糊也是复原处理的任务。这些模糊来自透镜散焦，相对运动，大气湍流，以及云层遮挡等。这些干扰可用维纳滤波、逆滤波、同态滤波等方法加以去除。去除图像畸变则需要借助图像的空间变换操作。

（5）图像重建

几何处理、图像增强、图像复原都是从图像到图像的处理，即输入的原始数据是图像，处理后输出的也是图像，而重建处理则是从数据到图像的处理，也就是说输入的是某种数据，而处理结果得到的是图像。该处理的典型应用就是 CT 技术，CT 技术发明于 1972 年，早期为 X 射线（X-ray）CT，后来发展的有 ECT、超声 CT、核磁共振（NMR）等。图像重建的主要算法有代数法、迭代法、傅立叶反投影法、卷积反投影法等，其中以卷积反投影法运用最为广泛，因为它的运算量小、速度快。值得注意的是三维重建算法发展得很快，而且由于与计算机图形学相结合，把多个二维图像合成三维图像，并加以光照模型和各种渲染技术，能生成各种具有强烈真实感及纯净的高质量图像。三维重建的主要算法有线框法、表面法、实体法、彩色分域法等，这些算法在计算机图形学中都有详尽的介绍。三维重建技术也是当今颇为热门的虚拟现实和科学可视化技术的基础。

（6）图像编码

图像编码研究属于信息论中信源编码范畴，其主要宗旨是利用图像信号的统计特性及人类视觉的生理学及心理学特性对图像信号进行高效编码，即研究数据压缩技术，以解决数据量大的矛盾。一般来说，图像编码的目的有 3 个：①减少数据存储量；②降低数据率以减少传输带宽；③压缩信息量，便于特征抽取，为识别做准备。就编码而言，Kunt 提出第一代、第二代编码的概念。Kunt 把 1948～1988 年 40 年中研究的以去除冗余为基础的编码方法称为第一代编码，如 PCM、DPCM、△M、亚取样编码法，变换编码中的 DFT、DCT、Walsh-Hadamard 变换等方法以及以此为基础的混合编码法均属于经典的第一代编码法。而第二代编码方法多是 20 世纪 80 年代以后提出的新的编码方法，如金字塔编码法、Fractal 编码、基于神经元网络的编码方法、小波变换编码法、模型基编码法等。现代编码法呈现出以下特点：①充分考虑人的视觉特性；②恰当地考虑对图像信号的分解与表述；③采用图像的合成与识别方案压缩数据率。

图像编码是经典的研究课题，经过 60 多年的研究已有多种成熟的方法得到应用。随着多媒体技术的发展，已有若干编码标准由 ITU-T 制定出来，如 JPEG、H.261、H.263、MPEG-1、MPEG-2、MPEG-4、MPEG-7、JBIG（Joint Bi-level Image Coding Expert Group，联合二值图像编码专家组）等。相信在未来会有更多、更有效的编码方法问世，以满足多媒体信息处理及通信的需要。

（7）模式识别

模式识别是数字图像处理的又一研究领域。当今，模式识别方法大致有 3 种，即统计识

别法、句法结构模式识别法和模糊识别法。

统计识别法侧重于特征，句法结构识别侧重于结构和基元，而模糊识别法是把模糊数学的一些概念和理论用于识别处理。在模糊识别处理中充分考虑人的主观概率，同时也考虑了人的非逻辑思维方法及人的生理和心理反映，这一独特性的识别方法目前正处于研究阶段，方法尚未成熟。

（8）图像理解

图像理解是由模式识别发展起来的方法。该处理输入的是图像，输出的是一种描述。这种描述并不仅是单纯的用符号做出详细的描绘，而且要利用客观世界的知识使计算机进行联想、思考及推论，从而理解图像所表现的内容。图像理解有时也叫景物理解。在这一领域还有相当多的问题需要进行深入研究。

以上所述的 8 项处理任务是图像处理所涉及的主要内容。其中，模式识别和图像理解涉及的是图像处理的更高级的技术，而本书侧重介绍如何利用 MATLAB 提供的工具箱函数对现有图像进行相关处理操作，为此对这部分内容感兴趣的读者可参考其他相关图像处理书籍。总的说来，经过多年的发展，图像处理经历了从静止图像到活动图像，从单色图像到彩色图像，从客观图像到主观图像和从二维图像到三维图像的发展历程。特别是与计算机图形学的结合后，图像处理已能产生高度逼真、非常纯净和更有创造性的图像。由此派生出来的虚拟现实技术的发展或许将从根本上改变我们的学习、生产和生活方式。

1.2.4 数字图像处理的应用

数字图像处理的应用越来越广，已经渗透到工程、工业、医疗保健、航空航天、军事、科研、安全保卫等各个方面，在国民经济各领域中发挥越来越大的作用。图像处理的具体应用领域可粗略概括为表 1-1。

表 1-1　图像处理的应用领域

学　科	应　用
物理、化学	结晶分析、谱分析等
生物、医学	细胞分析、染色体分类、X 射线成像、CT 等
环境保护	水质及大气污染调查等
地质	资源勘测、地图绘制、GIS 等
农业、林业	农产物估计、植被分布调查等
渔业	鱼群分布调查等
气象	卫星云图分析等
通信	传真、电视、多媒体通信等
工业	工业探伤、机器人、产品质量监测等
军事	导弹导航、军事侦察等
法律	指纹识别等

应用的典型例子有以下几个方面。

（1）遥感

在遥感的发展中，我们可以看到大量的与图像处理密切相关的技术。从世界上出现第一张照片（1839 年），意大利人乘飞机拍摄了第一张航空照片（1909 年），前苏联（1957 年）

及美国（1958 年）发射第一颗人造地球卫星等都为遥感技术的发展奠定了坚实的基础。1962 年国际上正式使用遥感一词（Remote Sensing）。此后，美国相继发射多颗陆地资源探测卫星（1972 年，LANDSAT-I，地面分辨率 59m×79m；1975 年，LANDSAT-II；1978 年，LANDSAT-III，地面分辨率 40m×40m；1982 年，LANDSAT-IV，地面分辨率 30m×30m，在这颗卫星上配置了 GPS 系统（Global Positioning System），定位精度在地心坐标系中为±10m。）

遥感图像处理的用处越来越大，效率及分辨率也越来越高，并广泛应用于土地测绘、资源调查、气象监测、环境污染监测、农作物估产和军事侦察等。当前，在遥感图像处理中主要面临的是数据量大和处理速度慢的矛盾。

（2）医学应用

图像处理在医学界的应用非常广泛，无论是在临床诊断还是病理研究都大量采用图像处理技术。它的直观、无创伤、安全方便的优点受到普遍的欢迎与接受。其应用的例子很多，如 X 射线照片的分析、血球计数与染色体分类等。目前广泛应用于临床诊断和治疗的各种成像技术，如超声波诊断等都用到图像处理技术。有人认为计算机图像处理在医学上应用最成功的例子就是 X 射线 CT（X-ray Computed Tomography）。1968～1972 年英国的 EMI 公司的 Hounsfeld 研制了头部 CT，1975 年又研制了全身 CT。20 世纪 70 年代下半叶美、日、法、荷相继生产出了 CT。其中主要研制者 Hounsfeld（英）和 Commack（美）获得了 1979 年的诺贝尔生理医学奖。这足以说明 CT 的发明与研究对人类的贡献之大、影响之深。类似的设备目前已有多种，如核磁共振 CT（NMRI，Nuclear Magnetic Resonance Imaging）、电阻抗断层图像技术（EIT，Electrical Impedance Tomography）和阻抗成像（Impedance Imaging），这是利用人体组织的电特性（阻抗、导纳、介电常数）形成人体内部图像的技术。由于不同组织和器官具有不同的电特性，这些电特性包含了解剖学信息。更重要的是人体组织的电特性随器官功能的状态而变化，因此，EIT 可望绘出反应与人体病理和生理状态相应功能的图像。目前，EIT 已发展了一些相应的算法（如图像重建算法），在临床应用中也正在探索（如神经中枢系统、呼吸系统、心血管系统、消化系统）。当前的主要问题是分辨能力差，原因是入射电流进入人体组织后呈三维分布发散，因此，指向性不强，并且电流在人体组织中的分布规律复杂，未知因素多。虽然 EIT 分辨率不高，但是生物阻抗技术提取的组织和器官的电特性信息对血液、气体、体液和不同组织成分有独特的鉴别能力，对血液的流动分布，肺内的气血交换，体液含量与流动等非常敏感，以此为基础，可进行心、脑、肺及相关循环系统的功能评价及血液动力学与流变学的研究。该技术对肺癌的早期发现显示出很大的优越性，这一点是现有的其他成像技术无法比拟的。

（3）通信

图像通信按业务性能划分可分为电视广播（点对面通信）、传真、可视电话（点对点通信）、会议电视（点对多点）、图文电视、可视图文以及电缆电视等。如按图像变化性质分，可分为静止图像通信和活动图像通信。

从历史上看，早在 1865 年就在法国试验成功传真通信（巴黎至里昂），但后来由于技术及经济原因发展一直非常缓慢。20 世纪 70 年代后，图像通信逐渐成为人们生活中常用的通信方式，随着大规模集成电路的发展，图像通信中所需的关键技术逐步得到解决，推动了图像通信的发展。1980 年 CCITT 为三类传真机和公众电话交换网上工作的数字传真建立了国际标准，1984 年 CCITT 提出了 ISDN 的建议，以及当今基于 IP 的多媒体通信都意味着非话业务通信方式已在通信中占有重要位置。图像通信主要有如下一些内容。

① 电视广播。单色电视广播 1925 年在英国实现，1936 年 BBC 开始电视广播。目前彩色电视有 3 种制式，即 NTSC（美国、日本等）、PAL（中国、西欧、非洲等）和 SECAM（法国、俄罗斯等）。

② 可视电话和会议电视。1964 年美国国际博览会展出了 Picture-phone MOD-I 可视电话系统，带宽为 1MHz。目前的可视电话/会议电视均采用数字压缩技术，也出现了相应的国际标准。如图像编码标准 H.261、H.263 等，会议电视的 H.230 标准，在专用通信网中用 PCM 一次群传输，速率为 2048kbit/s。桌面型系统遵循 H.323 标准。

③ 传真。传真是把文字、图表、照片等静止图像通过光电扫描的方式变成电信号加以传送的设备。1980 年 CCITT 为三类传真机和公众电话交换网上工作的数字传真建立了国际标准，即一类机——不压缩，4 线/mm，A4 文件传 6min；二类机——采用频带压缩技术（残留边带传输），4 线/mm，传 A4 文件需 3min；三类机——在传送前采用去冗余技术，在电话线上以 1min 传 A4 文件；四类机——在三类机的基础上发展的采用去冗余技术的传真设备，采用去冗余、纠错码技术在公众数据网上使用的设备，加 Modem 也可以在公众电话网上使用。经过多年发展，传真技术不断进步，现在已有仅数秒钟就可传送一幅 A4 文件的传真机，分辨率高达 16 点/mm。

④ 图文电视和可视图文。图文电视（Teletext）和可视图文（Videotext）是提供可视图形文字信息的通信方式。图文电视是单向传送信息，它是在电视信号消隐期发送图文信息，用户可用电视机和专用终端收看该信息；可视图文是双向工作方式，用户可用电话向信息中心提出服务内容或从数据库中选择信息。

⑤ 有线电视（CATV）。通过电缆或光缆传送的电视节目。第一个电缆电视系统于 1949 年安装在美国，采用光缆实现的 CATV 是 1977 年以后的事情。

（4）工业生产的质量控制

在生产线中对生产的产品及部件进行无损检测也是图像处理技术的一个广泛的应用领域。如食品包装出厂前的质量检查，浮法玻璃生产线上对玻璃质量的监控和筛选，甚至在工件尺寸测量方面也可以采用图像处理的方法加以自动实现。

（5）安全保障等方面的应用

该领域可采用模式识别等方法实现监控、指纹档案、案件侦破、交通管理等。

（6）教学和科研领域

教学及科研领域中也将大量应用图像处理技术。如科学可视化技术、远程培训及教学也将大量使用图像处理技术的成果。

（7）电子商务

在当前呼声甚高的电子商务中，图像处理技术也大有可为，如身份认证、产品防伪及数字水印技术等。

总之，图像处理技术应用还可列出相当多的领域，它在国家安全、经济发展、日常生活等方面充当越来越重要的角色，对国计民生的作用不可低估。

1.3 MATLAB 简介

MATLAB 语言是由美国 Mathworks 公司推出的计算机软件，经过多年的逐步发展与不断

完善，现已成为国际公认的最优秀的科学计算与数学应用软件之一，其内容涉及矩阵代数、微积分、应用数学、有限元法、科学计算、信号与系统、神经网络、小波分析及其应用、数字图像处理、计算机图形学、电子线路、电机学、自动控制与通信技术、物理、力学和机械振动等方面。MATLAB 的特点是语法结构简单，数值计算高效，图形功能完备，特别受到以完成数据处理与图形图像生成为主要目的技术研发人员的青睐。各国的高校学生（包括硕士生与博士生）也将 MATLAB 作为必须掌握的基本程序设计语言。

1.3.1 MATLAB 的发展简史

MATLAB 是 Matrix Laboratory（矩阵实验室）的缩写，最初由美国 Cleve Moler 博士在20世纪70年代末讲授矩阵理论和数据分析等课程时编写的软件包由LinPack与EisPack组成，旨在使应用人员免去大量经常重复的矩阵运算和基本数学运算等繁琐的编程工作。1984 年，Cleve Moler 博士和一批数学家、软件专家组建了 Mathworks 公司，开发出了第 2 代 MATLAB 软件，并推向市场。其内核改用 C 语言编写，提高了速度，另外还增加了绘图功能，使数值计算结果可以直接在 MATLAB 环境下用曲线和曲面等形式表示出来。1990 年，Mathworks 公司推出了以框图为基础的控制系统仿真工具 Simulink，它方便了系统的研究和开发，使控制工程师可以直接构造系统框图进行仿真，并提供了控制系统中常用的各种环节的模块库。1993 年，Mathworks 公司推出的 MATLAB 4.0 版在原来的基础上又做了较大改进，并推出了 Windows 版，使命令执行和图形绘制可以在不同窗口进行。1994 年推出了 MATLAB 4.2 版，从而得到了广泛的重视和应用。1999 年 1 月推出了 MATLAB 5.3 版本，真正实现了 32 位运算，其速度更快、功能更完善、界面更友好，并且提供了 Internet 搜索引擎，可以协助用户寻求在线帮助。新版本 6.0（Release 12）、6.1（Release 13）又作了更精细的改进，6.1 版于 2001 年 5 月推向市场。相对于 5.3 版本，新版本增加了许多新的功能，主要表现在以下几个方面。

- 增强了用户界面的交互性，其窗口界面更加友好。
- 增加了工具箱的种类，增强了工具箱的功能。
- 增加了许多功能函数。
- 扩充了绘图功能。
- 增强了对多维矩阵、稀疏矩阵的运算功能。
- 增加了微分方程的解法和积分方程的算法。
- 扩充了矢量和矩阵的类型。
- 提供了更新的和完备的在线帮助文档。
- 提供了输入数据向导（import wizard）。

现在，又进一步推出了 MATLAB7.0 版本，界面操作更加简单，工具箱函数进一步丰富，程序执行更加高效。在 1.3.3 节将会对 MATLAB7.0 版本的新特点给予较为详细的介绍。

1.3.2 MATLAB 的特点

MATLAB 之所以成为世界流行的科学计算与数学应用软件，是因为它有着下列强大的功能。

- 高质量、强大的数值计算功能

为满足复杂科学计算任务的需要，MATLAB 汇集了大量常用的科学和工程计算方法，从各种函数到复杂运算，包括矩阵求逆、矩阵特征值、奇异值、工程计算函数以及快速傅立叶变换等。MATLAB 强大的数值计算功能是其优于其他数学应用软件的重要原因。

- 数据分析和科学计算可视化功能

MATLAB 不但科学计算功能强大，而且在数值计算结果的分析和数据可视化方面也远远优于其他同类软件。在科学计算和工程应用中，经常需要分析大量的原始数据和数值计算结果，MATLAB 能将这些数据以图形的方式显示出来，使数据间的关系清晰明了。

- 强大的符号计算功能

科学计算有数值计算与符号计算两种，在数学、应用科学和工程计算领域，常常会遇到符号计算问题，仅有优异的数值计算功能并不能解决科学计算时的全部需要。在 MATLAB 的发展过程中，Mathworks 公司从 Waterloo 大学购买了 Maple 的使用权，并以 Maple 的核心部分作为其符号计算功能的引擎，依靠 Maple 已有的库函数，实现了 MATLAB 环境下符号计算功能。

- 强大的非线性动态系统建模和仿真功能

MATLAB 提供了一个模拟动态系统的交互式程序 Simulink，允许用户通过绘制框图来模拟一个系统，并动态地控制该系统。Simulink 能处理线性、非线性、连续、离散等多种系统，它包括应用程序扩展集 Simulink Extensions 和 Blocksets。其中 Simulink Extensions 是支持在 Simulink 环境下进行系统开发的一些工具类应用程序，如 Simulink Accelerator、Real Time Workshop 及 Stateflow；而 Blocksets 则是针对 DSP（数字信号处理）、Communications（通信）、Nonlinear control Design（非线性控制设计）、Fixed Point（定点）等几个特殊应用领域设计的程序的集合。

- 灵活的程序接口功能

应用程序接口（API）是一个允许用户编写的与 MATLAB 互相配合的 C 或 Fortran 程序的文件库。MATLAB 提供了方便的应用程序接口，用户可以在 MATLAB 环境下直接调用已经编译过的 C 和 Fortran 子程序，在 MATLAB 和其他应用程序之间建立客户机/服务器关系。同样，在 C 和 Fortran 程序中，也可以调用 MATLAB 的函数或命令，使得这些语言可以充分利用 MATLAB 的矩阵运算功能和方便的绘图功能。

- 文字处理功能

MATLAB 记事本成功地将 MATLAB 与文字处理系统 Microsoft Word 集成为一个整体，为用户进行文字处理、科学计算、工程设计创造了一个统一的工作环境。用户不仅可以利用 Word 的文字编辑处理功能，方便地创建 MATLAB 的系统手册、技术报告、命令序列、函数程序、注释文档以及与 MATLAB 有关的教科书等 6 种文档，而且还能从 Word 访问 MATLAB 的数值计算和可视化结果。

另外，MATLAB 还具有支持科学计算标准的开放式可扩充结构和跨平台兼容的特点，能够很好地解决科学和工程领域内的复杂问题。MATLAB 的技术特点主要表现在以下几个方面。

（1）界面友好、编程效率高。MATLAB 是一种以矩阵为基本变量单元的可视化程序设计语言，语法结构简单，数据类型单一，命令表达方式接近于常用的数学公式。

这使 MATLAB 用户在短时间内就能快速掌握其主要内容和基本操作。MATLAB 不仅能

免去大量的经常重复的基本数学运算，而且其编译和执行速度都远远超过了采用 C 和 Fortran 语言设计的程序。可以说，MATLAB 在科学计算与工程应用方面的编程效率远远高于其他高级语言。

（2）功能强大，可扩展性强。MATLAB 语言不但提供了科学计算、数据分析与可视化、系统仿真等强大的功能，而且还具有可扩展性特征。Mathworks 公司针对不同领域的应用，推出了自动控制、信号处理、图像处理、模糊逻辑、神经网络、小波分析、通信、最优化、数理统计、偏微分方程、财政金融等 30 多个具有专门功能的 MATLAB 工具箱。各种工具箱中的函数可以互相调用，也可以由用户更改，这一点读者可以在今后的使用中慢慢体会，这是非常有用的。MATLAB 支持用户对其函数进行二次开发，用户的应用程序可以作为新的函数添加到相应的工具箱中。

（3）图形功能灵活方便。MATLAB 具有灵活的二维与三维绘图功能，在程序的运行过程中，用户可以方便迅速地用图形、图像、声音、动画等多媒体技术直接表述数值计算结果，可以选择不同的坐标系，可以设置颜色、线型、视角等，还可以在图中加上比例尺、标题等标记，在程序运行结束后改变图形标记、控制图形句柄等，并且还可以将图形嵌入到用户的 Word 文件中。

（4）在线帮助，有利于自学。用户可以借助于 MATLAB 环境下的“在线帮助”学习各种函数的用法及其内涵。对于 MATLAB6.0 以上版本，还可以用 HTML 方式查询更为详细的参考资料。另外还可以直接访问 Mathworks 公司的网站，以获得常见问题解答（FAQ）、产品指南和 MATLAB 书籍等更丰富的帮助信息。

总之，MATLAB 语言已经成为科学计算、系统仿真、信号与图像处理的主流软件。

1.3.3 MATLAB7.0 的新增功能

MATLAB7.0 在以前版本的基础上，进行了完善和扩充，在开发环境、数学运算、编程和数据类型、外部接口、图形和用户接口创建方面增加了新功能。

1. 开发环境

- MATLAB 可以运行于 Windows、Linux、Solaris 和 Macintosh 平台下。
- 桌面上增加了 Start 按钮，可以快速访问所有工具，这是 MATLAB7.0 版本新增的按钮。
- 在命令窗口菜单 Edit 项里还增加了 Find 对话框，可以在命令窗口、历史命令窗口、当前 M 文件目录、所选择文件的当前目录、整个 MATLAB 路径中搜索该关键字。
- 打印功能。选 File→Page Setup 选项，可选择打印命令窗口的某一部分，以及设置字体等。
- 对于命令窗口的优选项，增加了 Keyboard 和 Indenting 优选项。Command line Key bindings 有两个选项——Emacs（MATLAB standard）和 Windows。如果选择 Emacs，快捷键 Ctrl+F 的功能是光标向前移动一个字符；而选择 Windows，Ctrl+F 的功能是打开 Find 对话框。在此菜单中，对于历史命令窗口和编辑器增加了自动存盘选项。
- 在命令窗口，选择文本，然后单击鼠标右键并选择 Open Selection 选项可以在工作区中打开一个变量，或者在编辑窗口打开一个文件或者函数。
- 编辑器可以显示行数、列数和当前函数。

- 新增加了一些文件系统操作函数，如 movefile、ZIP 函数和 E-mail 函数 sendmail 等。

2. 数学运算

- 可以解算常数延迟项的微分方程。
- 可以解决单边值 ODE 求解问题，实现此功能的函数是 bvp4c。
- 体积积分问题，即可以求三重积分，实现此功能的函数是 triplequad。
- 求解 gamma 函数的对数导数，实现此功能的函数是 psi。

3. 编程和数据类型

- MATLAB 的 JIT 加速器。加快了 M 文件应用中函数和脚本的执行速度。性能加速部分增加了如何充分利用 JIT 加速器和怎样利用 MATLAB Profiler 优化性能等内容。
- 支持规则的表达式。利用 regexp、regexpi 和 regexprep 函数实现对规则表达式进行字符查找和替换。
- 增加了一些新函数。这些函数涉及错误产生、规则表达式、排序、整数变化和进行文件操作等 17 个函数。
- 增加了新的警告和句柄属性。error 和 warning 函数的参数可以是一个或者更多，如，error('File %s not found', filename)，
  warning('Ambiguous parameter name, "%s".', param)。
- 动态的结构字段名，即矩阵、单元阵列和结构，在运行时可以变化。
- 增加了两个新的“与”和“或”操作符，即&&和||，用于计算混合逻辑表达式。
- 为 ismember 函数增加了一个新的输出。语法：[tf, index] = ismember(A,S,…)，index 表示 S 中包含 A 的最大索引值。
- 增加了 true 和 false 函数。
- 循环中的中断。可以利用快捷键 Ctrl+C 中断循环，返回到命令窗口。
- 修改了 copyfile、mfilename 和 mkdir 函数。
- 支持 64 位整数，并增加了 64 位文件处理函数。
- 使用了新的 MATLAB 定时器对象。
- 增强了音频功能。

4. 图形

- 增加了 Colormap editor，可以编辑图像的色彩。
- 增加了新的文本属性，可以控制文本的背景色彩。

5. 外部接口

- 增加了几个新的 MATLAB 接口函数。
- 具有更可靠的存储器管理功能。
- 增加了事件句柄的灵活性。
- 有可列举的属性值，且可以增加用户常用属性。
- 图形属性和方法接口。
- 可以自动改变文件标签，可以更新 CALLBACK 和 M 文件代码。

- FILE-EXPORT，无需 FIG 文件就可以把 GUI 导出为一个单个的 M 文件。
- 工具条上的 MATLAB Editor 图标使得访问编辑器更加容易。

6. 创建 GUI

- 新的结构用于产生 M 文件，更易于理解和编程。
- GUIDE 快速启动对话框和模板。

1.4 MATLAB 用于图像处理

MATLAB 中的基本数据结构是由一组有序的实数或复数元素构成的数组，同样地，图像对象的表达采用的是一组有序的灰度或彩色数据元素构成的实值数组。MATLAB 中通常用二维数组来存储图像，数组的每一个元素对应于图像的一个像素值。例如，由 200 行和 300 列的不同颜色点组成的一幅图像在 MATLAB 中采用 200×300 的矩阵存储。在下面的介绍中可以看到 MATLAB 支持多种类型的图像，而不同类型的图像其存储结构通常是不同的。如 RGB 图像则需要一个三维数组，3 个数据维分别对应于某像素点的红色、绿色和蓝色强度值。由于对图像采用了通用的数据矩阵的表达方式，MATLAB 中原有的所有基本矩阵操作都可应用于图像矩阵，例如，我们要查看图像 I 中某像素点的强度值，可以采用类似的表达方式：$I(x, y)$，它代表了图像 I 的第 x 行和第 y 列的像素值。

1.4.1 MATLAB7.0 图像处理工具箱

数字图像研究的领域非常广泛，从学科上可以分为图像的数字化、图像变换、图像增强、图像恢复、图像分割、图像分析和理解，以及图像的压缩等。

MATLAB 图像处理工具箱提供了丰富的图像处理函数，主要可以完成以下功能：

- 图像的几何操作；
- 图像的邻域和图像块操作；
- 线性滤波和滤波器设计；
- 图像变换；
- 图像分析和增强；
- 二值图像形态学操作；
- 图像复原；
- 图像编码；
- 感兴趣区域处理。

MATLAB 图像工具箱提供的函数大多数是 M 文件，我们可以查看这些文件的代码并进行改进，也可以把自己编写的代码加入其中，来扩充图像处理的功能。

1.4.2 MATLAB7.0 支持的图像文件格式

表 1-2 列出了 MATLAB7.0 支持的图像文件格式。

表 1-2　　　　**MATLAB7.0 支持的图像文件格式**

文件格式	文件格式全名	变量		
'bmp'	Windows Bitmap (BMP)	1bit, 4bit, 8bit, 16bit, 24bit 以及 32bit 非压缩图像；4bit 和 8bit RLE(run-length encoded）图像		
'cur'	Windows Cursor resources (CUR)	1bit, 4bit 和 8bit 非压缩图像		
'gif'	Graphics Interchange Format (GIF)	1bit, 4bit 和 8bit 图像		
'hdf'	Hierarchical Data Format (HDF)	8bit 带或不带关联颜色映射表的光栅图像数据集，24bit 光栅图像数据集		
'ico'	Windows Icon resources (ICO)	1bit, 4bit 和 8bit 非压缩图像		
'jpg'或'jpeg'	Joint Photographic Experts Group (JPEG)	图像类型	位数	压缩性
		灰度图	8bit 或 12bit	有损
		灰度图	8bit, 12bit 或 16bit	无损
		RGB	24bit 或 36bit	有损或无损
'pbm'	Portable Bitmap (PBM)	二进制编码或 ASCII 编码的 1bit 图像		
'pcx'	Windows Paintbrush (PCX)	1bit, 8bit 和 24bit 图像		
'pgm'	Portable Graymap (PGM)	每个像素有任意位数的 ASCII 编码图像或直到 16bits 的二进制编码图像		
'png'	Portable Network Graphics (PNG)	1bit, 2bit, 4bit, 8bit 和 16bit 灰度图像；8bit 和 16bit 索引值图像；24bit 和 48bit RGB 图像		
'pnm'	Portable Anymap (PNM)	3 个 Portable Bitmap 系列图像格式的通用名称：Portable Bitmap（PBM），Portable Graymap（PGM）和 Portable Pixel Map（PPM）		
'ppm'	Portable Pixmap (PPM)	每个颜色成分有任意位数的 ASCII 编码图像或直到 16bit 的二进制编码图像		
'ras'	Sun Raster (RAS)	1bit bitmap, 8bit 索引值图像，24bit 真彩色图像和有 alpha 数据的 32bit 真彩色图像		
'tif' or 'tiff'	Tagged Image File Format (TIFF)	1bit, 8bit 和 24bit 非压缩图像；1bit, 8bit 和 24bit packbits 压缩图像；1bit CCITT 压缩图像；16bit 灰度图像；16bit 索引值图像；48bit RGB 图像		
'xwd'	X Windows Dump (XWD)	1bit, 8bit ZPixmaps 和 XYBitmaps；1bit XYPixmaps		

1.4.3　MATLAB 中图像的数据存储类型及其转换

在默认的情况下，MATLAB 将图像中的数据存储为双精度类型（double），即 64 位的浮点数。这种存储方法的优点在于使用中不需要数据类型转换。因为几乎所有的 MATLAB 及其工具箱函数都可使用 double 作为参数类型。然而对于图像存储来说，这种数据表达方式并不总是合适。用 64 位来表示图像数据会导致巨大的存储量。例如，一幅大小为 1000×1000 的图像拥有 1000 000 个像素点，如果按照 double 类型存储该图像数据，则需要大约 8MB 的存储空间。当然，对于更大的图像，所需的存储空间将更加巨大。因此，为了应对图像处理中对存储量的需要，提高系统运行效率，MATLAB 还支持图像数据的其他类型——8 位和 16 位无符号整型数组的存储类型，这样将只需 double 类型数组的 $\frac{1}{8}$ 或 $\frac{1}{4}$ 的存储量。在 MATLAB 中，数据矩阵中包含 uint8 数字类型的图像称之为 8 位图；同理，数据矩阵中包含 uint16 数字类型的图像称为 16 位图。MATLAB 及工具箱中的大多数操作及函数都不支持 uint8 类型和 uint16 类型，uint8 和 uint16 的优势仅仅在于节省存储空间，在涉及运算时要将其转换成 double 型。

由于在 MATLAB 中图像存储和函数运算的数据格式要求不尽相同，因而通常需要进行图像数据存储格式的转换。在 MATLAB7.0 中用于存储类型转换的函数有 im2double，im2uint8

和 im2uint16。这些函数可以自动处理原始图像数据的偏移。例如，函数 im2uint8 是用于将像素值取值区间为[0,1]的 double 类型的 RGB 图像转换为取值区间为[0,255]的 uint8 类型的 RGB 图像，函数使用方式为：RGB2 = im2uint8(RGB1)。

对于图像存储类型的转换过程，有所不同的是索引值图像，这是由于索引值图像对应像素值表示的是其颜色映射表的序号值，而并非是像素点的真实亮度值，直接按上述方式转换将会造成不可预知的错误。MATLAB7.0 工具箱中的函数 imapprox 正是为了解决这一问题而提出的。将 uint16 或 double 类型的索引值图像转换成 uint8 类型，由于 uint8 数组只有 256 个离散值，imapprox 函数在执行这一类型转换时，首先对颜色映射表进行操作，减少图像颜色的数目，然后基于新的颜色映射表进行转换。

利用 MATLAB 提供的 image 函数，可以直接显示 8 位图像或 16 位图像，而不必将其转换为双精度浮点类型，然而，当图像是 uint8 或 uint16 类型时，MATLAB 对矩阵值的解释有所不同，这主要依赖于图像的具体类型。下面简单介绍一下 8 位和 16 位的索引色图像、灰度图像和 RGB 图像。

- 8 位和 16 位索引色图像

如果图像数据矩阵的类型是 unit8 或 unit16，则其值在作用于颜色映射表的索引之前，必须进行值为 1 的偏移，即值 0 指向矩阵 map 的第一行，值 1 指向第二行，依此类推，并且因为 image 函数自动提供了这种偏移，所以不管图像数据矩阵是双精度浮点类型，还是 uint8 或 uintl6 显示的方法均相同。

但是，当用户在进行类型转换时，由于偏移量的存在，必须将 uint8 或 uint16 的数据加 1，然后才能将其转换为双精度浮点类型。例如：

X64=double(X8)+1;

或

X64=double(X16)+l;

相反，若将双精度浮点类型转换为 uint8 或 uint16 类型的数据时，在转换之前，必须将其减去 1。例如：

X8=uint8(X64)-1;

或

X16=uint16(X64)-1;

- 8 位和 16 位灰度图像

在 MATLAB 中，双精度浮点类型的图像数组中的数值的取值范围通常为[0，1]，而 8 位和 16 位无符号整数类型的图像的取值范围分别为[0，255]和[0，65535]。

- 8 位和 16 位 RGB 图像

8 位 RGB 图像的颜色数据是[0，255]之间的整数，而不是[0，1]之间的浮点值。因此，在 8 位 RGB 图像中，颜色值为（255，255，255）的像素显示为白色。不管 RGB 图像是何种类型，MATLAB 都通过以下的代码来显示，即 image(RGB);

将 RGB 图像从双精度的浮点类型转换为 uint8 无符号整数类型时，首先要乘以 255，即：

RGB8=uint8(round(RGB64*255));

相反，如果将 uint8 无符号整数类型的 RGB 图像转换为双精度的浮点类型时，首先要除以 255，即 RGB64=double(RGB8)/255;

另外，如果将 RGB 图像从双精度的浮点类型转换为 uin16 无符号整数类型时，必须乘以

65535，即 RGB16=uint16(round(RGB64*65535));

同样，如果将 uintl6 无符号整数类型的 RGB 图像转换为双精度浮点类型时，首先要除以 65535，即 RGB64=double(RGB16)/65535。

1.4.4 MATLAB7.0 支持的图像类型及其转换

MATLAB 中，一幅图像可能包含一个数据矩阵，也可能包含一个颜色映射表矩阵。MATLAB 中有 4 种基本的图像类型：索引色图像、灰度图像、RGB 图像和二值图像。此外，MATLAB 还支持由多帧图像组成的图像序列。下面具体介绍 MATLAB 如何描述这几种图像。

- 索引色图像

索引色图像包括一个数据矩阵 X，一个颜色映像矩阵 Map。其中 Map 是一个包含 3 列和若干行的数据阵列。Map 矩阵的每一行分别表示红色、绿色和蓝色的颜色值。在 MATLAB 中，索引色图像是从像素值到颜色映射表值的“直接映射”。像素颜色由数据矩阵 X 作为索引指向矩阵 Map 进行索引。例如，值 1 指向矩阵 Map 中的第一行，2 指向第二行，依此类推。

颜色映射表通常和索引色图像存储在一起。当用户在调用函数 imread 时，MATLAB 自动将颜色映射表与图像同时加载。在 MATLAB 中可以选择所需要的颜色映射表，而不必局限于使用默认的颜色映射表。我们可以使用属性 CDataMapping 来选取其他的颜色映射表，包括用户自定义的颜色映射表。

- 灰度图像

MATLAB 中，一幅灰度图像是一个数据矩阵 I，其中 I 的数据均代表了在一定范围内的颜色灰度值。MATLAB 把灰度图像存储为一个数据矩阵，该数据矩阵中的元素分别代表了图像中的像素。矩阵中的元素可以是双精度的浮点数类型、8 位或 16 位无符号的整数类型。大多数情况下，灰度图像很少和颜色映射表一起保存。但是在显示灰度图像时，MATLAB 仍然在后台使用系统预定义的默认的灰度颜色映射表。

- RGB 图像

RGB 图像，即真彩色图像，在 MATLAB 中存储为 $n\times m\times 3$ 的数据矩阵。数组中的元素定义了图像中每一个像素的红、绿、蓝颜色值。需要指出的是，RGB 图像不使用 Windows 颜色映射表。像素的颜色由保存在像素位置上的红、绿、蓝的强度值的组合来确定。图像文件格式把 RGB 图像存储为 24 位的图像，红、绿、蓝分别占 8 位。这样可以有约 1 000 万种颜色（即 2^{24}=16 777 216）。

MATLAB 的 RGB 数组可以是双精度的浮点数类型、8 位或 16 位无符号的整数类型。在 RGB 的双精度型数组中，每一种颜色是用在 0 和 1 之间的数值表示。例如，颜色值（0，0，0）的像素显示的是黑色；颜色值(1，1，1)的像素显示的是白色。每一像素的 3 个颜色值保存在数组的第三维中。例如，像素(10，5)的红、绿、蓝颜色值分别保存在三维矩阵中的元素 RGB(10，5，1)、RGB(10，5，2)、RGB(10，5，3)中。

- 二值图像

与灰度图像相同，二值图像只需要一个数据矩阵，每个像素只取两个灰度值。二值图像可以采用 uint8 或 double 类型存储，工具箱中以二值图像作为返回结果的函数都使用 uint8 类型。

- 图像序列

MATLAB 的图像处理工具箱还支持将多帧图像连接成图像序列。图像序列是一个四维的数组，图像帧的序号在图像的长、宽、颜色深度之后构成第四维。比如一个包含了 5 幅 400×300 像素的真彩色图像序列，其大小为 400×300×3×5。

许多图像处理工作都对图像类型有特定的要求。比如要对一幅索引色图像滤波，首先必须将它转换成真彩色图像，否则结果是毫无意义的。在 MATLAB 中，各种图像类型之间的转换关系如图 1-2 所示。

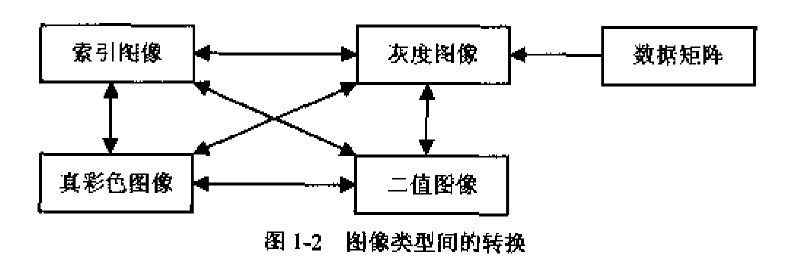


图 1-2　图像类型间的转换

MATLAB 的图像处理工具箱提供了许多图像类型转换函数，以实现上述的各种转换，下面就具体介绍这些函数的用法和功能。

- dither 函数

该函数通过颜色抖动（颜色抖动即改变边沿像素的颜色，使像素周围的颜色近似于原始图像的颜色，从而以空间分辨率来换取颜色分辨率）来增加输出图像的颜色分辨率，从而实现转换图像。

该函数的调用格式如下：

X=dither(RGB，map)

BW=dither(I)

X=dither(RGB，map)表示将真彩色图像 RGB 按指定的颜色映射表 map 抖动成索引色图像 X。

BW=dither(I)表示将灰度图像 I 抖动成二值图像 BW。

另外，输入图像可以是 double 或 uint8 类型，输出图像如果是二值图像或颜色种类不超过 256 的索引色图像，则是 uint8 类型，否则为 double 型。

- gray2ind 函数

该函数的功能是将灰度图像转换成索引色图像。其调用格式如下：

[X，map]=gray2ind(I，n)

[X，map]=gray2ind(I，n)表示按指定的灰度级数 n 将灰度图像 I 转换成索引色图像 X，n 的默认值为 64。

另外，输入图像 I 可以是 double 类型、uint8 类型或 uint16 类型。如果颜色映射表的长度小于等于 256，则输出图像是 uint8 类型，否则为 double 类型。

下面的例程把一幅灰度图像转换为一幅索引色图像。

- grayslice 函数

该函数的功能是通过设定阈值将灰度图像转换成索引色图像。其调用格式如下：

X=grayslice(I，n)

X=grayslice(I，v)

X=grayslice(I，n)将灰度图像 I 均匀量化为 n 个等级，然后转换为索引色图像 X。

X=grayslice(I，v)按指定的阈值矢量 v（其每一个元素都在 0 和 1 之间）对图像 I 的值域进行划分，而后转换成索引色图像 X。

另外，输入图像 I 可以是 double 或 uint8 类型。如果阈值数量小于 256，则返回图像 X 的数据类型是 uint8，X 的值域为[0，n]或[0，length(v)]。否则，返回图像 X 为 double 类型，值域为[1，n+1]或[1，length(v)+1]。

- im2bw 函数

该函数的功能是通过设置亮度阈值将真彩色、索引色、灰度图像转换成二值图像。该函数的调用格式如下：

BW=im2bw(I，level)

BW=im2bw(X，map，level)

BW=im2bw(RGB，level)

BW=im2bw(I，level)、BW=im2bw(X，map，level)和 BW=im2bw(RGB，level)分别表示将灰度图像、索引色图像和真彩色图像二值化为图像 BW。level 是归一化的阈值，取值在[0,1]之间。

另外，输入图像可以是 double 或 uint8 类型，输出图像为 uint8 类型。

- ind2gray 函数

该函数的功能是将索引色图像转换成灰度图像。其调用格式如下：

I=ind2gray(X，map)

另外，输入图像可以是 double 或 uint8 类型，输出图像为 double 类型。

- ind2rgb 函数

该函数的功能是将索引色图像转换成真彩色图像。其调用格式如下：

RGB=ind2rgb(X, map)

另外，输入图像可以是 double 或 uint8 类型，输出图像为 double 类型。

- mat2gray 函数

该函数的功能是将一个数据矩阵转换成一幅灰度图像。其调用格式如下：

I=mat2gray(A，[amin amax])

I=mat2gray(A)

I=mat2gray(A，[amin amax])表示按指定的取值区间[amin amax]将数据矩阵 A 转化为灰度图像 I，amin 对应灰度 0(最暗)，amax 对应 1(最亮)。如果不指定区间[amin amax]，则 MATLAB 自动将 A 阵中的最小元素设为 amin，最大元素设为 amax。

另外，输入矩阵 A 与输出图像 I 都为 double 类型。

- rgb2gray 函数

该函数的功能有两个：第一个功能是将一幅真彩色图像转换成一幅灰度图像；第二个功能是把一幅索引色图像的彩色颜色映射表转换为灰度颜色映射表。其调用格式如下：

I=rgb2gray(RGB)

newmap=rgb2gray(map)

I=rgb2gray(RGB)表示将真彩色图像 RGB 转换成灰度图像 I。

newmap = rgb2gray(map)表示将彩色颜色映射表 map 转化成灰度颜色映射表。需要注意的是，如果输入的是真彩色图像，则可以是 uint8 或 double 类型，输出图像 I 与输入图像类型相同。如果输入的是颜色映射表，则输入、输出都是 double 类型。

1.4.5 颜色空间

在 MATLAB 图像处理工具箱中，总是直接（RGB 图像）或者间接（索引色图像）地使用 RGB 数据来表示颜色。但是除了 RGB 颜色模型之外，还有许多别的颜色模型，例如 HSV 模型，这些不同的颜色模型叫做色彩空间。

颜色模型是三维颜色空间中的一个可见光子集，它包括某个颜色域的所有颜色。常用的颜色模型有 NTSC、HSV 和 YCbCr 模型等。

NTSC 模型广泛应用于美国等国家的电视信号。它的特点是信号的强度信息和颜色信息相分离，同一个信号可以方便地同时表示彩色图像和黑白图像。在 NTSC 格式中，图像由 3 个分量表示：亮度（luminance）用 Y 表示；色度（hue）用 I 表示；饱和度（saturation）用 Q 表示。第一个分量亮度 Y 表示灰度信息，后两个分量表示彩色信息。因此 NTSC 模型使用的是 Y-I-Q 色彩坐标轴，NTSC 模型的色彩空间又称 YIQ 空间。NTSC 模型与 RGB 模型的转换函数如下：

- rgb2ntsc

rgb2ntsc 函数用于将 RGB 模型转换成 NTSC 模型，其语法格式为：

yiqmap = rgb2ntsc(rgbmap)

YIQ = rgb2ntsc(RGB)

yiqmap = rgb2ntsc(rgbmap)将 RGB 空间中 $m\times3$ 的色彩表 rgbmap 转换为 YIQ 空间中的调色板 yiqmap。

YIQ = rgb2ntsc(RGB)将真彩色图像 RGB 转换为 YIQ 空间中的图像 YIQ。

- ntsc2rgb

ntsc2rgb 函数用于将 NTSC 模型转换为 RGB 模型，其语法格式如下：

rgbmap = ntsc2rgb(yiqmap)

RGB = ntsc2rgb(YIQ)

rgbmap=ntsc2rgb(yiqmap)将 YIQ 空间中 $m\times3$ 的调色板 yiqmap 转换成 RGB 空间中的色彩表 rgbmap。RGB=ntsc2rgh(YIQ)将 YIQ 色彩空间的图像 YIQ 转换为真彩色图像 RGB。

HSV 模型通常用于选择颜色，它是面向用户的一种复合主观感觉的色彩模型，比 RGB 模型更接近人们对颜色的感知。图 1-3 是 HSV 模型的示意图。

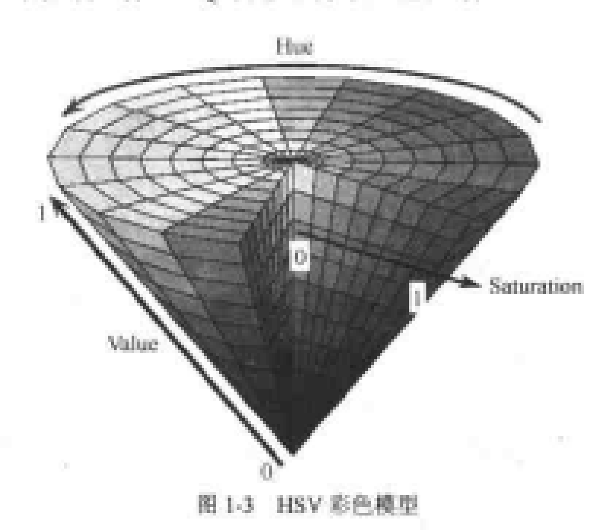


图 1-3　HSV 彩色模型

HSV 模型中 H 表示颜色的种类，由绕 V 轴的旋转角决定，"每一种颜色和它的补色之间相差 180°。S 为饱和度，V 为亮度。当色度由 0 变到 1，HSV 的颜色由红变为黄、绿、青、蓝、洋红，然后变回红色。当饱和度由 0 变到 1，颜色由不饱和变为饱和。当亮度由 0 变到 1，颜色越来越亮。HSV 模型与 RGB 模型的转换函数如下：

- rgb2hsv

rgb2hsv 函数用于将 RGB 模型转换成 HSV 模型，其语法格式为：

hsvmap=rgb2hsv(rgbmap)

HSV=rgb2hsv(RGB)

其中 hsvmap=rgb2hsv(rgbmap)将 RGB 空间中 $m\times3$ 的色彩表 rgbmap 转换成 HSV 色彩空间中的调色板 hsvmap。HSV=rgb2hsv(RGB)将真彩色图像 RGB 转换为 HSV 空间中的图像 HSV。

- hsv2rgb

hsv2rgb 函数用于将 HSV 模型转换为 RGB 模型，其语法格式为：

rgbmap=hsv2rgb(hsvmap)

RGB=hs2rgb(HSV)

其中 rgbmap=hsw2rgb(hsvmap)将 HSV 色彩空间的调色板 hsvmap 转换为 RGB 空间中的色彩表 rgbmap，rgbmap 和 hsvmap 都是 $m\times3$ 的矩阵。RGB=hsv2rgb(HSV)将 HSV 色彩空间的图像 HSV 转换为真彩色图像 RGB。

YCbCr 模型广泛应用于数字视频。在 YCbCr 模型中，Y 为亮度，Cb 和 Cr 共同描述图像的色调，其中 Cb、Cr 分别为蓝色分量和红色分量相对于参考值的坐标。

- rgb2ycbcr

rgb2Ycbcr 函数用于将 RGB 模型转换成 YCbCr 模型，其语法格式为：

ycbcrmap=rgb2ycbcr(rgbmap)

YCBCR=rgb2ycbcr(RGB)

这里 ycbcrmap=rgb2ycbcr(rgbmap)将 RGB 空间中的色彩表 rgbmap 转换成 YCbCr 空间中的调色板 ycbcrmap。YCBCR=rgb2ycbcr(RGB)将真彩色图像 RGB 转换为 YCbCr 空间中的图像 YCBCR。

- ycbcr2rgb

ycbcr2rgb 函数用于将 YCbCr 模型转换成 RGB 模型，其语法格式为：

rgbmap = ycbcr2rgb(ycbcrmap)

RGB = ycbcr2rgb(YCBCR)

rgbmap=ycbcr2rgb(ycbcrmap)将 YCbCr 空间中的调色板 ycbcrmap 转换成 RGB 空间中的色彩表 rgbmap。RGB=ycbcr2rgb(YCBCR)将 YCbCr 空间中的图像 YCBCR 转换为真彩色图像 RGB。

1.4.6 图像对象属性详解

MATLAB 中可以调用 image 函数和 imagesc 函数来创建图像对象。与线条对象、片块对象、表面对象以及文本对象类似，图像对象也是坐标轴对象的子对象。另外，与所有的句柄图形对象一样，图像对象也包含许多用户可以设置的以调整其屏幕显示外观的属性。其中最重要的属性包括 CData 属性、CDataMapping 属性、XData 属性和 YData 属性等。

- 图像对象的 CData 属性

图像对象的属性 CData 包含一个数据阵列。下面的代码中，H 表示由 image 函数创建的图像对象的句柄，X 和 Y 代表同一个矩阵。

H=image(X);

colormap(map);

Y=get(H, 'CData')

其中，属性 CData 数据阵列的维数决定了 MATLAB 是使用颜色映射表还是使用 RGB 真彩色的方式显示该图像。如果 CData 属性的数据阵列是一维的，则图像显示为索引色图像或灰度图像。但不管是何种方式，MATLAB 均使用颜色映射表。如果 CData 属性的数据阵列是三维的，形如 $m \times n \times 3$，那么 MATLAB 就将其显示为真彩色图像。

- 图像对象的 CDataMapping 属性

图像对象的 CDataMapping 属性决定了图像是索引色图像还是灰度图像。如果将该属性设置为"direct"，则该图像显示为索引色图像，在这种情况下，CData 属性中的数据阵列直接作为当前图形窗口的颜色映射表中的颜色索引。在 MATLAB 中，如果我们在调用 image 函数时只使用一个参数，则系统自动将 CDataMapping 属性的值设置为"direct"。

例如，我们在命令行中输入下面的代码：

```
H=image(X);
colormap(map);
get(H, 'CDataMapping');
```

回车执行后，得到下面的结果：

```
ans=
    direct
```

另外，如果将 CDataMapping 属性的值设置为"scaled"，MATLAB 将该图像对象显示为灰度图像。在这种情况下，MATLAB 将 CData 属性的数据线性缩放，生成颜色映射表的索引，其中缩放因子由坐标轴对象的 CLim 属性确定。

在 MATLAB 中，只要利用 imagesc 函数创建图像对象，系统自动将 CDataMapping 属性的值设置为"scaled"，并且同时调整其父坐标轴对象的 CLim 属性。

例如，在命令行中输入下面的代码：

```
H=imagesc(I, [0 1]);
colormap(map);
get(H, 'CDataMapping')
```

回车执行后，得到下面的结果：

```
ans=
    scaled
```

输入代码：

```
get (gca, 'CLim')
```

执行后，得到下面的结果：

```
ans=
    [0 1]
```

- 图像对象的 XData 和 YData 属性

图像对象的 XData 和 YData 属性决定了图像的坐标系统，对于一个数据矩阵为 $m \times n$ 的图像对象来说，其 XData 属性的默认值是[1 n]，YData 属性的默认值是[1 m]，这样设置的意义如下：

① 图像数据矩阵的左边的列具有一个值为 1 的 x 坐标。

② 图像数据矩阵的右边的列具有一个值为 n 的 x 坐标。

③ 图像数据矩阵的上边的行具有一个值为 1 的 y 坐标。

④ 图像数据矩阵的下边的行具有一个值为 m 的 y 坐标。

另外，用户还可以定义自己的坐标系来覆盖系统的默认的设置。

1.4.7 MATLAB7.0 程序与 VC 程序的对比

基本的图像处理操作一般包括读入图像、显示图像、处理图像和存储图像等几个部分。在下面的章节中，读者可以看到使用 MATLAB7.0 完成上述操作通常只需非常简单的几行代码即可实现，而使用 VC 编写的图像处理程序代码将会非常庞大。

下面举一个简单的例子，读者可以从中对比 VC 程序和完成同样功能的 MATLAB 代码在实现难度上的差异。例如，为了实现图像读取和显示操作，VC 程序代码需要完成的工作涉及到确定待图像文件，读取文件头信息，确定文件格式，设置调色板，顺序读取图像数据，然后调用系统提供的 API 函数，设定显示的各种参数，设计显示界面，接下来才可以看到一幅图像显示在显示屏上。尽管这是图像处理中最基本的步骤，但要实现这一过程要求程序员对待处理图像的文件格式信息、数据操作以及系统 API 函数有比较深入的了解，完全实现下来，不算上系统提供的 API 函数，全部代码也有几百行以上。而使用 MATLAB 提供的工具箱函数，读者可以编写下面的代码轻松地实现它。

```
I=imread('LENA256.bmp');
imshow(I)
```

如图 1-4，看看我们在屏幕上得到了什么？



图 1-4 图像读取和显示

是不是很轻松呢？使用 MATLAB 图像处理工具箱函数将大大减轻图像数据的繁杂操作，使我们更加快捷地实现图像处理任务，不再把很多的时间花在调试一些基本代码上，而是把更多的精力倾注于各种图像处理算法的效果上。

为什么会这么轻松呢？让我们看一下 Mathworks 公司的程序员为我们做了哪些工作。请在 MATLAB 命令视窗内键入以下命令，那么一切都明白了。

```
help imread.m
```

可以得到以下代码（去掉了其中的大部分注释行）：

```
function [X, map, alpha] = imread(varargin)
[filename, format, extraArgs, msg] = parse_inputs(varargin{:});
if (~isempty(msg))
      error('MATLAB:imread:inputParsing', '%s', msg);
end
% Download remote file.
if (strfind(filename, '://'))
      url = true;
```

```
    if (~usejava('mwt'))
        error('MATLAB:imread:noJava', 'Reading from a URL requires a Java…
                        Virtual Machine.')
    end
    try
        filename = urlwrite(filename, tempname);
    catch
        error('MATLAB:imread:readURL', 'Can''t read URL "%s".', filename);
    end
else
    url = false;
end
if (isempty(format))
    % The format was not specified explicitly.
    % Verify that the file exists.
    fid = fopen(filename, 'r');
    if (fid == -1)
        if ~isempty(dir(filename))
            error('MATLAB:imread:fileOpen', ['Can''t open file "%s" for…
                        reading;\nyou' ' may not have read permission.'], filename);
        else
            error('MATLAB:imread:fileOpen', 'File "%s" does not exist.', filename);
        end
    else
        % File exists.   Get full filename.
        filename = fopen(fid);
        fclose(fid);
    end
    % Try to determine the file type.
    format = imftype(filename);
    if (isempty(format))
        error('MATLAB:imread:fileFormat', 'Unable to determine the file format.');
    end
else
    % The format was specified explicitly.
    % Verify that the file exists.
    fid = fopen(filename, 'r');
    if (fid == -1)
        % Couldn't open using the given filename; search for a
        % file with an appropriate extension.
```

```
            fmt_s = imformats(format);
            if (isempty(fmt_s))
                error('MATLAB:imread:fileFormat', ['Couldn''t find format %s in the ···
                            format registry.'' See "help imformats".'], format);
            end
            for p = 1:length(fmt_s.ext)
                fid = fopen([filename '.' fmt_s.ext{p}]);
                if (fid ~= -1)
                    % The file was found.   Don't continue searching.
                    break
                end
            end
        end
        if (fid == -1)
            if ~isempty(dir(filename))
                error('MATLAB:imread:fileOpen', ['Can''t open file "%s" for ···
                            reading;\nyou' ' may not have read permission.'], filename);
            else
                error('MATLAB:imread:fileOpen', 'File "%s" does not exist.', filename);
            end
        else
            filename = fopen(fid);
            fclose(fid);
        end
    end
    % Get format details.
    fmt_s = imformats(format);
    % Verify that a read function exists
    if (isempty(fmt_s.read))
        error('MATLAB:imread:readFunctionRegistration', 'No reading function for format ···
                    %s.   See "help imformats".', fmt_s.ext{1});
    end
    if ((fmt_s.alpha) && (nargout == 3))
            % Use the alpha channel.
        [X, map, alpha] = feval(fmt_s.read, filename, extraArgs{:});
    else
        % Alpha channel is not requested or is not applicable.
        alpha = [];
        [X, map] = feval(fmt_s.read, filename, extraArgs{:});
    end
```

```
% Delete temporary file from Internet download.
if (url)
    delete_download(filename);
end
%%%
%%% Function delete_download
%%%
function delete_download(filename)
try
    delete(filename);
catch
    warning('MATLAB:imread:tempFileDelete', 'Can''t delete temporary file "%s".',...
                            filename)
end
%%%
%%% Function parse_inputs
%%%
function [filename, format, extraArgs, msg] = ...
        parse_inputs(varargin)
filename = '';
format = '';
extraArgs = {};
msg = '';
% Parse arguments based on their number.
switch(nargin)
case 0
    % Not allowed.
    msg = 'Too few input arguments.';
    return;
case 1
    % Filename only.
    filename = varargin{1};
otherwise
    % Filename and format or other arguments.
    filename = varargin{1};
    % Check whether second argument is a format.
    if (ischar(varargin{2}))
        fmt_s = imformats(varargin{2});
    else
        fmt_s = struct([]);
```

```
        end
        if (~isempty(fmt_s))
            % The argument matches a format.
            format = varargin{2};
            extraArgs = varargin(3:end);
        else
            % The argument begins the format-specific parameters.
            extraArgs = varargin(2:end);
        end
    end
```

MATLAB 已经为我们实现了繁杂的各种函数，当使用时，我们只需调用它，并简单设定几个参数，就大功告成。以上代码，就是函数 imread 的实现代码，它替我们做了全部工作，使得我们可以通过简单的函数调用就可以完成需要的功能。读者可以仔细查看这段代码，其中有很多是来自 MATLAB 自身的功能函数，可见 MATLAB 已经把很多复杂的数据操作隐藏起来了，通过这种方式，在完成同样功能时，MATLAB 代码比 VC 代码要简单一些。

由以上对比可以发现，使用 MATLAB 工具箱函数的确可以简化操作，减轻编程压力。当然 MATLAB 编程一直以来也有其不足的地方，就是运算效率不如 VC 实现的代码，但这个问题正随版本的不断升级而得到很大程度的改善，想必用过 MATLAB7.0 的读者对这一观点有着同感。另外，MATLAB 编程有一个很大的优点，就是重复编译效率很高。这些读者可以在接下来的使用过程中慢慢体会。

1.5 本章小结

本章对图像处理的重要地位、涉及的处理任务和 MATLAB 工具箱的支持进行了介绍，并简单回答了对使用 MATLAB 语言表示图像及进行图像处理所涉及的一些基本问题。在后面的章节中，将会具体针对不同的图像处理任务，向读者介绍图像处理工具箱中各种函数的应用，并注重图像处理的实例介绍和操作结果的对比。

第 2 章　MATLAB7.0 的图像文件操作

MATLAB7.0 图像处理工具箱为用户提供了丰富的函数，使得基本的图像文件操作如读取、写入文件以及显示变得非常容易。这些操作也是进行其他图像处理工具箱函数学习前必须了解的。

本章内容分为 3 个部分：2.1 节介绍了图像文件的读写操作，涉及到工具箱函数 imread、infinfo 和 imwrite 的使用；2.2 节介绍了两种图像显示方法，主要涉及到函数 imshow 和 imview 的使用；2.3 节介绍了几种特殊的图像显示技术。

2.1　图像文件的读写

MATLAB 为用户提供了专门的函数以从图像格式的文件中读写图像数据。这种方法不像其他编程语言需要编写复杂的代码，只需要简单地调用 MATLAB 提供的函数即可。MATLAB 支持的图像文件格式有*.cur、*.bmp、*.hdf、*.ico、*.jpg、*.pcx、*.png、*.tif 和*.xwd。用于图像文件读写相关操作的工具箱函数有 imread、imfinfo 和 imwrite。下面分别介绍其使用方法。

2.1.1　图像文件的读取

MATLAB 中利用函数 imread 来实现图像文件的读取操作。其格式主要有以下几种类型。

（1）A = imread(filename,fmt)

该语句用于读取字符串“filename”指定的灰度图像或彩色图像，“fmt”指定了文件的格式。如果该图像文件不在当前目录下或是 MATLAB 路径的目录下，需要指定图像文件在系统中的完整文件路径。fmt 的可能取值参考第 1 章中给出的 MATLAB 所能支持的图像文件格式。如果 imread 函数在指定的路径下不能找到 filename 指定的图像文件，会试图寻找图像文件“filename.fmt”。

imread 函数返回数组 A 表达的图像数据，如果读取的是灰度图像，则 A 是一个 $m \times n$ 的二维数组。类似地，如果读取的是彩色图像，则 A 表示一个 $m \times n \times 3$ 的三维数组。数组的数据类型由图像文件的数据类型决定。对大多数文件格式，彩色图像数据使用 RGB 颜色空间类型。另外，也可以返回 CIELAB、ICCLAB 或 CMYK 等颜色空间的数据类型。

（2）[X,map] = imread(filename,fmt)

可以看到，相比上一条语句，该语句等号的左边多了一个参数，用于读取索引色图像，其中 X 用于存储索引色图像数据，即是相关颜色映射表的序号值，map 用于存储与该索引色图像相关的颜色映射表。filename 和 fmt 的意义同前，不再赘述。

（3）[…] = imread(filename)

该语句在执行图像读取任务时，首先需要从图像文件 filename 的内容推断其图像类型，即并不显式地给出图像类型 fmt，而是推断得到。该语句左边的“[…]”表示根据待读取的图像数据是真实像素值或索引色图像的相应颜色映射表的序号值而分别采用如语句（1）和语句

（2）的不同形式。

（4）[…] = imread(URL,…)

该语句用于读取引自 Internet URL 的图像，URL 要求必须包含协议类型（protocol type），如 http://。该语句中 imread 函数的第二个参数即是所要读取的 Internet URL。语句左边的形式同语句（3）。

imread 函数的格式除以上介绍的比较常用的 4 种类型外，还有针对某些特殊类型的图像读取语句采用的其他参数，想要进一步了解可以参考 MATLAB 的帮助文档。

在使用 imread 函数执行图像读取任务时，除注意其使用格式外，还应注意其针对不同数据存储类型图像的不同操作。

通常情况下，我们通过 imread 函数读取的大多数图像都是 8 位的。当把这些图像加载到内存中时，MATLAB 就将其存储为 uint8 类型。

此外，MATLAB 还支持 16 位的 PNG 和 TIFF 图像。当用户读取这类图像时，MATLAB 就将其存储为 uint16 类型。而对于索引图像来说，图像矩阵的本身为 uint8 或 unit16，imread 函数仍将颜色映射表读取并存储在一个双精度浮点类型的矩阵中。

2.1.2 图像文件的写入

MATLAB 中利用函数 imwrite 来实现图像的文件写入操作，与 imread 函数的作用相对。其语句格式通常有以下几种：

imwrite(A, filename, fmt)

imwrite(X, map, filename, fmt)

imwrite(…,filename)

imwrite(…,Param1,Val1,Param2,Val2…)

（1）imwrite(A, filename, fmt)

该语句执行的操作是把图像数据 A 写入到由 filename 指定的输出文件中去，存储的格式由 fmt 指定。与 imread 函数的使用类似，如果所指定的输出文件 filename 不在当前目录下或 MATLAB 的目录下，必须指明其完整路径；fmt 的取值参考 MATLAB7.0 所支持的图像文件格式列表（第 1 章）。

A 不能为空，对于灰度图像来说，A 是一个 $m \times n$ 的数组，对于彩色图像来说，A 是一个 $m \times n \times 3$ 的三维数组。如果 fmt 指定格式为 TIFF，则 imwrite 函数可以接受 $m \times n \times 4$ 的三维数组。

（2）imwrite(X, map, filename, fmt)

该语句用于写入索引色图像，其中 X 表示索引色图像数据数组，map 表示其关联的颜色映射表，filename 为 fmt 格式的输出文件。如果 X 是 uint8 或 uint16 类型的数组，imwrite 将数组中的实际数据写入到文件 filename 中。如果 X 是 double 类型的数组，imwrite 函数采用 uint8(X−1) 表示数组中的值并写入到相应的输出文件中。参数 map 必须是有效的 MATLAB7.0 颜色映射表。

（3）imwrite(…,filename)

该语句在写入图像到文件时，从 filename 的扩展名推断图像的文件格式，该扩展名要求必须是 MATLAB7.0 所支持的类型值。imwrite 函数在 filename 之前的参数与前面提到的语句相同的使用方式。

（4）imwrite(…,Param1, Val1, Param2, Val2…)

该语句用于指定 HDF、JPEG、PBM、PGM、PNG、PPM 和 TIFF 等类型输出文件的不同参数。例如，当写入一个 HDF 文件时，可以指定输出图像的质量（Quality）、压缩性（Compression）和写入模式（WriteMode），而写入 JPEG 文件时，则可以指定像素位数（BitDepth）、注释（Comment: Empty or not）、模式（Mode: lossy or lossless）和质量（Quality）等，关于其他类型图像与格式相关的参数设置可参考帮助文档中的相关列表。

另外，需要注意的是，当利用 imwrite 函数保存图像时，MATLAB 默认的储存方式是将其简化为 uint8 的数据类型。与读取图像文件类型类似，MATLAB 在文件保存时还支持 16 位的 PNG 和 TIFF 图像。因此，当用户保存这类文件时，MATLAB 就将其存储为 uint16 类型。

2.1.3 图像文件信息的查询

在 MATLAB7.0 中，可以使用 imfinfo 函数查询图像文件的信息。其语句格式如下：

info=imfinfo('文件名'，文件格式）

info=imfinfo('文件名')

由该函数获取的信息依赖于文件类型的不同而不同，但至少应包含以下一些内容：

'文件名'（文件名）；

FileModDate（文件最后一次的修改时间）；

FileSize（文件的大小，单位为字节）；

Format（文件格式）；

FormatVersion（文件格式的版本号）；

Width（图像的宽度，单位为像素）；

Height（图像的高度，单位为像素）；

BitDepth（每个像素的位数）；

ColorType（图像类型，即是 RGB 图像、灰度图像还是索引色图像）。

例如，为了查询图像文件 woman2-512x512.tif 的信息，可以在 MATLAB 的 Command Window 中键入以下代码：

```
info=imfinfo('woman2-512x512.tif')
```

执行后，得到的结果如下：

```
info =
              Filename: 'woman2-512x512.tif'
           FileModDate: '04-Apr-2001 15:35:34'
              FileSize: 262314
                Format: 'tif'
         FormatVersion: []
                 Width: 512
                Height: 512
              BitDepth: 8
             ColorType: 'grayscale'
       FormatSignature: [77 77 0 42]
             ByteOrder: 'big-endian'
```

```
NewSubfileType: 0
 BitsPerSample: 8
   Compression: 'Uncompressed'
PhotometricInterpretation: 'BlackIsZero'
  StripOffsets: 154
SamplesPerPixel: 1
  RowsPerStrip: 512
StripByteCounts: 262144
   XResolution: 0.0139
   YResolution: 0.0139
ResolutionUnit: 'None'
      Colormap: []
PlanarConfiguration: 'Chunky'
     TileWidth: []
    TileLength: []
   TileOffsets: []
TileByteCounts: []
   Orientation: 1
     FillOrder: 1
GrayResponseUnit: 0.0100
MaxSampleValue: 255
MinSampleValue: 0
  Thresholding: 1
```

另外，在图像处理中，经常需要查看图像的直方图统计特性，为了完成这一操作，MATLAB 工具箱提供了函数 imhist。如下面的实现代码：

```
I=imread('LENA256.bmp');
imshow(I);
imhist(I)
```

执行结果如图 2-1 所示。



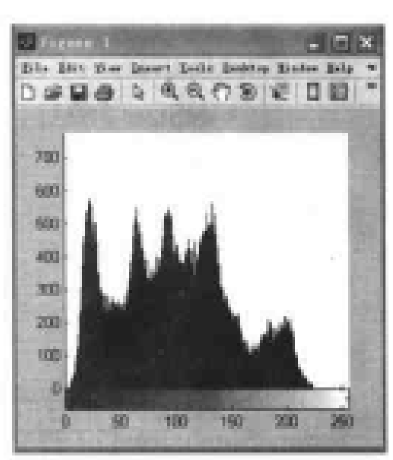


图 2-1　查看图像的直方图

2.2 图像文件的显示

在 MATLAB7.0 中用于显示图像的方式有两种，一种是使用 MATLAB 图像浏览器(Image Viewer)，通过调用 imview 函数实现，其界面如图 2-2 所示；另一种是使用 MATLAB 的通用图形图像视窗，通过调用 imshow 函数实现，其界面如图 2-3 所示。下面就图像的两种显示方式分别进行介绍。

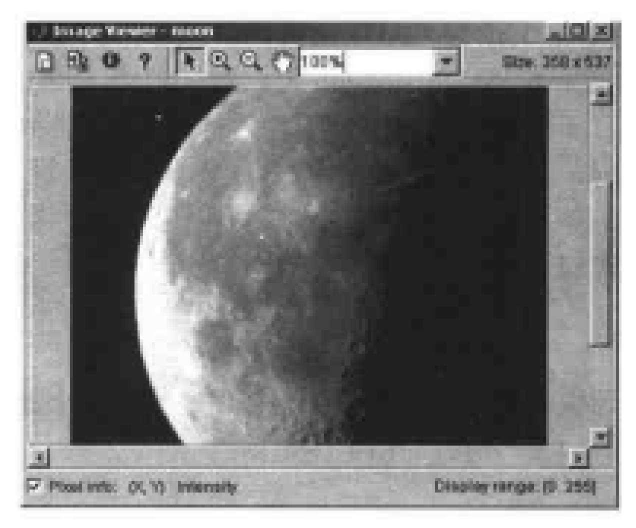


图 2-2 MATLAB 图像浏览器（Image Viewer）



图 2-3 MATLAB 通用图像视窗

2.2.1 使用图像浏览器显示图像

这一部分将介绍如何用图像浏览器来显示图像，主要从以下几个方面帮助读者学会如何使用图像浏览器。

- 图像浏览器的打开与关闭；
- 图像浏览器导航功能；
- 像素区域工具；
- 图像信息工具。

（1）图像浏览器的打开与关闭

- 在图像浏览器中显示图像

当需要打开图像浏览器时，可以调用 imview 函数，并指定想要用浏览器浏览的图像。利用以下语句命令行，就可以在 MATLAB 工作窗口中显示一幅图像：

```
moonfig = imread('moon.tif');
imview(moonfig);
```

也可以直接指定图像名，语句格式如下：

```
imview('moon.tif');
```

当然，该图像文件必须位于 MATLAB 的当前路径。需要注意的一点是，这种直接指定

图像文件名的显示方式不同于上一种方式，该语句执行后并不意味着该图像被存储到了MATLAB 工作空间（workspace）里。

如果指定的文件包括多幅图像，通常 imview 将只会显示文件中的第一幅图像文件。为了显示文件中的多幅文件，可以用 imread 函数现把每一幅图像读入 MATLAB 的 workspace 中，然后调用多次 imview 来显示每一幅图像。如果想要同时显示多帧图像的所有帧，可以考虑使用函数 montage，具体方法将在 2.3 节介绍。

- 指定图像的起始大小

在默认的情况下，imview 将 100%地显示整幅图像。在本文中，100%意味着 imview 函数将是图像的每一像素一一映射到屏幕窗口的每一像素。在某些情况下，特别是在使用一些小图像时，这就需要指定图像的大小。为了控制图像起始在浏览器中显示的大小，可通过下面几种方法来改变：设置函数 imview 的 IntialMagnification 值，默认值为 100，表示按照原图像 100%的放大倍数显示，设置为“fit”表示按照图像浏览器窗口全屏显示；在 imview 函数中使用 InitialMagnification 参数，该参数的设置将会屏蔽 ImviewIntialMagnification 的作用，从而在调用 imview 函数时可以不必再考虑 ImviewIntialMagnification 的设置。其设置格式如下：

```
imshow(X, map, 'InitialMagnification', 'fit')
```

- 关闭图像浏览器

要关闭当前图像浏览器，可以直接单击该图像浏览器窗口上的关闭按钮。当有多个浏览器窗口同时处于打开状态时，可以使用下面的语句关闭所有的图像浏览器：

```
imview close all
```

另外，还有一种关闭的方法，就是借助使用 imview 函数时返回的指向图像浏览器的指针完成关闭操作。

（2）图像浏览器的基本功能

图像浏览器可以在分离的窗口中显示一幅图像并提供图像的大小信息、图像像素值的范围和鼠标所在位置的像素值。除此之外，他还提供了其他 3 个工具：全景查看窗口（实现导航功能）、像素区域工具和图像信息窗口。图 2-4 显示了图像浏览器和他的辅助工具。

- 全景查看窗口

如果一个图像很大或者是采用很大的放大倍数来显示，则图像浏览器窗口内将只能显示整幅图像的一部分，类似于通常的 Windows 窗口，此时将会引入滚动条来拖动窗口内的图像以达到查看全部图像的目的。然而，有时这种方法不见得能够满足用户的需要，而可能会产生盲人摸象的效果。图像浏览器提供了一个方便的工具，即全景查看窗口，如图 2-4 中左侧的独立小窗口。

要想激活全景查看窗口，可以使用鼠标单击图像浏览窗口工具栏上的按钮。该工具采用一个小的独立窗口（全景查看窗口）显示整幅图像，在该窗口中，可以看到一个矩形框——细节矩形（detail rectangle），在图像浏览器的主窗口中显示的图像就是该矩形框选中的部分图像。当然，当图像窗口中显示的是整幅图像时，在全景查看窗口中将不会显示细节矩形。通过移动该矩形框就可以达到在主图像浏览器的主窗口中显示图像不同部分的目的，即实现了图像的导航功能，而与此同时，又可以通过全景查看窗口了解显示部分在整幅图像中的位置，使用户可以在把握图像整体的同时查看到图像的细节。

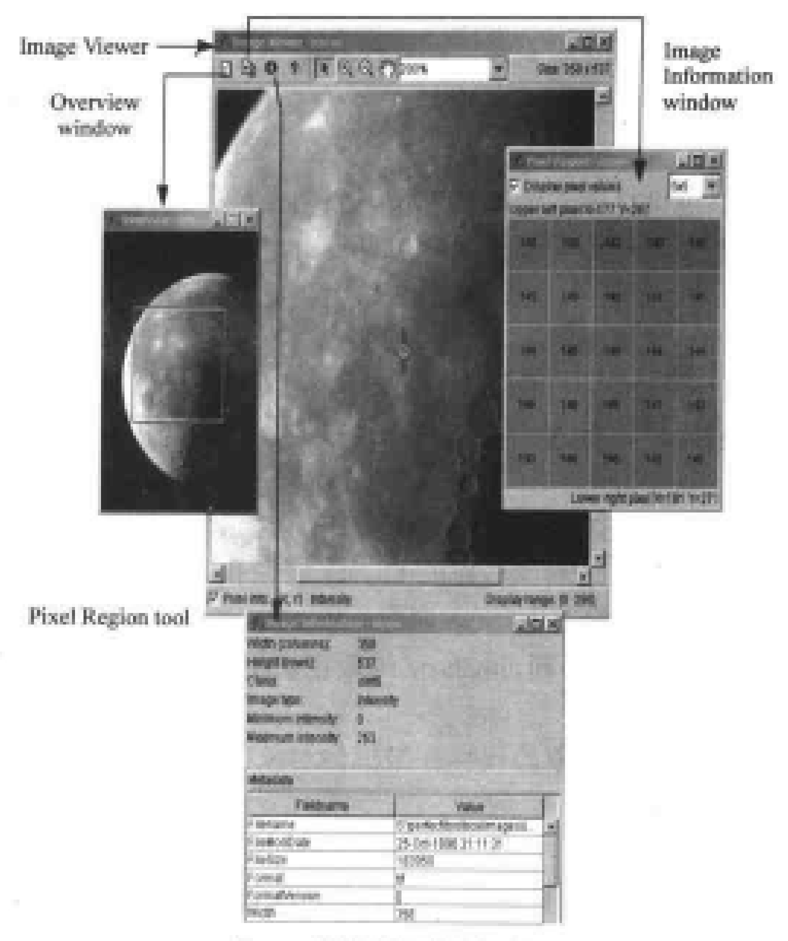


图 2-4　图像浏览器的基本功能

另外，图像浏览器还提供了几个按键实现导航功能，使用户可以很容易地实现一幅图像的导航功能。这些按钮包括窗口拖动工具、放大缩小工具和图像比例放大工具。其图标见图 2-5 所示。

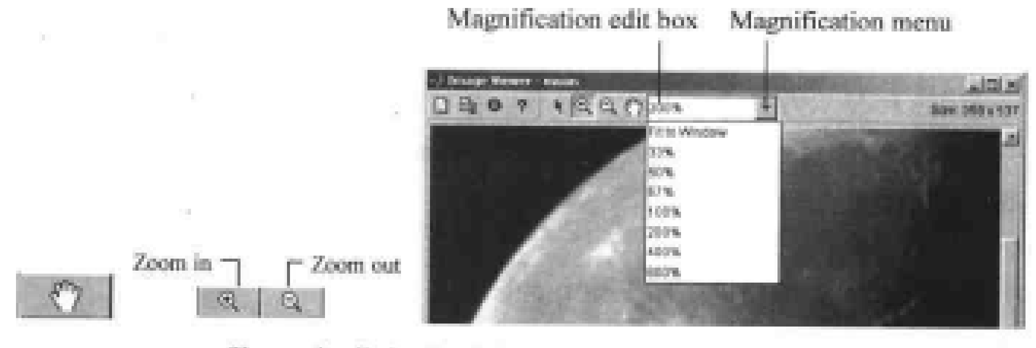


图 2-5　窗口拖动工具、放大缩小工具和图像比例放大工具

- 像素区域工具

该工具用来检查图像特定区域的像素值，当我们使用这一功能时，像素区域工具将会在一个独立的窗口内显示选定区域的像素值。我们可以通过拖动图像上的像素区域矩形来选择不同的区域，这个工具的使用使得我们定量认识图像中特定元素变得更加容易。

相对于图像导航浏览功能，像素区域工具可以提供图像中一些特别像素的信息。要激活像素区域工具，可以单击图像浏览器窗口工具栏上的按钮，在图像浏览器主窗口的图像上

会出现一个矩形光标✢，称为像素区域矩形。矩形中的像素值将显示在另外一个分离的特别窗口中，这样可以很容易知道图像各部分的具体像素值。如图 2-4 右侧独立窗口所示，其中像素区域矩形的大小有多种选择，如 5×5、6×6 等，读者可根据需要自行设定。

- 图像信息窗口

图像信息工具可以提供显示在图像浏览器中的图像信息。它提供的信息与 imfinfo 函数提供的信息相同，只是更直观。为了查看这些信息，可以单击图像浏览器工具栏中的按钮 i，这样图像的信息就会显示在另一个分离的小窗口中，显示的内容包括图像的宽和高、图像类型、颜色类型和密度的最大值/最小值等。

2.2.2 使用 imshow 函数显示图像

当用户使用 imshow 函数显示一幅图像时，该函数将自动设置图像窗口、坐标轴和图像属性。这些自动设置的属性包括图像对象的 CData 属性和 CDataMapping 属性、坐标轴对象的 CLim 属性、图像窗口对象的 Colormap 属性。

另外，根据用户使用参数的不同，imshow 函数在调用时除了完成前面提到的属性设置外，还可以完成下面的操作：

（1）设置其他的图形窗口对象的属性和坐标轴对象的属性以优化显示效果。如可以设置隐藏坐标轴及其标示等。

（2）包含和隐藏图像边框。

（3）调用 truesize 函数来设定图像到屏幕像点的映射关系。

imshow 函数的调用格式如下：

```
A = imread('filename.fmt');
imshow(A);
```

另外也可以采用另一种方式，即

```
imshow(' filename.fmt');
```

这种显示方式要求被显示的图像要么在当前目录下或 MATLAB 的目录下，要么就必须指定该图像的完整路径。

注意：使用该种显示方式并没有将图像数据存储在 MATLAB 的 workspace 中。如果想把当前显示的图像存储到 workspace 中，必须借助 getimage 函数，该函数将返回当前句柄图形图像对象的数据，使用格式为：

```
moon = getimage;
```

该语句将把当前显示的图像赋给变量 moon。

与函数 imview 的使用类似，在调用 imshow 函数在图形图像视窗内显示图像时，既可以使用默认的显示设置，即一个图像像素对应一个屏幕像点，也可以通过设置 imshow 函数的参数来达到更改图像显示方式的目的，此时需要借助 truesize 函数来设定图像像素到屏幕像点的映射关系。

函数 imshow 的调用格式通常有以下几种形式：

```
imshow(I,n)
imshow(I,[low high])
imshow(BW)
```

imshow(X,map)

imshow(RGB)

imshow(···,display_option)

imshow(x,y,A,···)

imshow filename

'\ = imshow(···)

imshow(I, n)表示用 n 个离散的灰度级显示图像 I。如果省略了 n，imshow 函数将使用 24 位表示的 256 个灰度级来显示图像。有些系统则默认使用 64 个灰度级。

例如显示一幅图像，使用语句：

lena=imread('LENA256.bmp');

imshow(lena,256)

得到如图 2-6 所示的运行结果。

imshow(I,[low high])表示把图像 I 作为一幅灰度图像来显示，[low high]指定了图像 I 的数据范围。图像中所有灰度级不超过 low 的像素显示为黑色，灰度级不低于 high 的像素显示为白色，灰度级处于限定范围内的像素按照其原来的灰度级显示。如果限定范围为空，imshow 函数默认的获取 low 和 high 的值分别为 min(I(:))和 max(I(:))，显示原则同上。

例如，执行下面的语句

imshow(lena,[50, 150])

得到如图 2-7 所示的显示结果。



图 2-6　256 个灰度级的图像



图 2-7　按灰度级范围[50, 150]修正的图像

而执行语句 imshow(lena,[])时，由于图像 lena 自身就是 256 级灰度图像，因此得到的显示结果同图 2-6 所示。

imshow(BW)用于显示二值图像 BW，即显示 0 为黑色，1 为白色。显示效果如图 2-8 所示。

说明：如果该二值图像矩阵属于 double 类，则 imshow 函数将其视为灰度图来对待，同时将 CDataMapping 的属性值设置为 scaled，CLim 的属性值设置为[0,1]；Colormap 颜色

映射表的属性值设置为灰度颜色映射表。因此，在这种情况下，图像数据矩阵中值 0 所对应的像素显示为黑色，而值 1 所对应的像素显示为白色。



图 2-8 二值图像

imshow(X, map)用于显示索引色图像 X，map 是与其相关的颜色映射表。

imshow(RGB)用于显示真彩色图像 RGB。

imshow(…,display_option)显示图像，“…”表示根据显示任务的不同可以采取上面介绍的某种格式。display_option 选取 'notruesize' 或 'truesize'，用于指定图像显示时图像像素点与屏幕像点的映射关系。

imshow(x, y, A,…)用 2 个元素的矢量 x 和 y 建立非默认的空间坐标系统，x 和 y 指定了 MATLAB 句柄图形图像对象（Handle Graphics image object）属性 XData 和 YData.。

imshow filename 可以直接显示文件 filename 中的图像。

h = imshow(…)，h 表示被显示图像的句柄。

2.3 特殊的图像显示技术

除了前面已经介绍的 imshow 函数以外，MATLAB 的图像处理工具箱还提供了一些可以实现特殊显示功能的函数。下面就具体介绍这些函数的用法和功能。

2.3.1 添加颜色条

在 MATLAB 的图像显示中，可以利用 colorbar 函数将颜色条添加到坐标轴对象中。添加进来的颜色条用来指示图像中不同颜色所对应的数据值。

举例说明 colorbar 函数的语法格式。在这个例子中，定义了灰度级的 uint8 类型的图像被滤波，滤波函数为 filter2，关于滤波我们在后续的章节中会介绍，然后调用 imshow 显示图像，并调用 colorbar 函数设置颜色条。函数代码如下：

```
RGB = imread('saturn.png');
I = rgb2gray(RGB);
h = [1 2 1; 0 0 0; -1 -2 -1];
I2 = filter2(h,I);
imshow(I2,[]), colorbar('vert')
```

执行结果如图 2-9 所示。

另外，当我们通过调用函数 imshow 在通用图像视窗显示图像时，也可以利用视窗上的工具条直接添加颜色条。

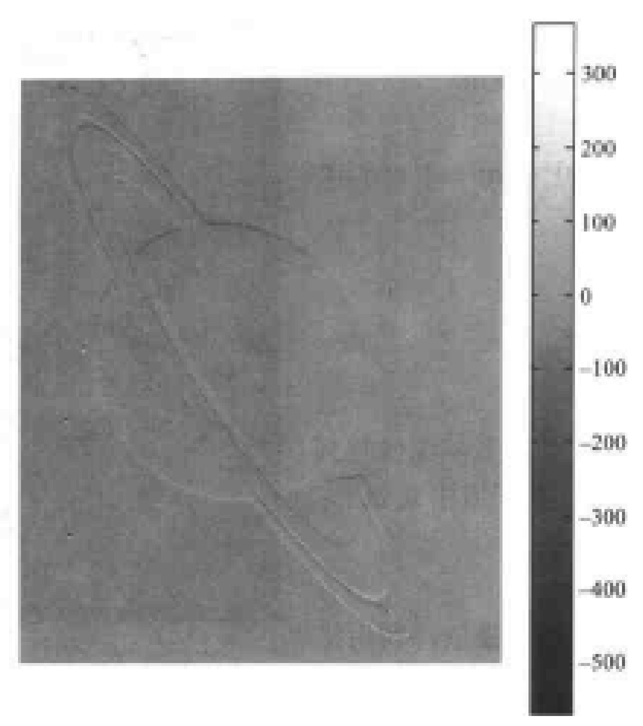


图 2-9 添加颜色条示意图

2.3.2 显示多帧图像阵列

在 MATLAB 中，在多帧图像阵列中查看图像，有下面几种方式：

- 独立显示每一帧，调用 imshow 函数。
- 同时显示所有的帧，调用 montage 函数。
- 将多帧图像阵列转换为电影动画，调用 immovie 函数。

下面将具体介绍这几个函数的用法。

（1）单帧显示

下面通过一个例程来说明图像的单帧显示。

```
%调用 cat 函数将两幅灰度图像合并成一个具有两帧的图像阵列，然后再调用
%imshow 函数来显示第二帧图像
lena=imread('LENA256.bmp');
girl=imread('Girl.bmp');
A=cat(3, lena, girl);
imshow(A(:,:,2))
```

得到如图 2-10 所示的结果。



图 2-10 图像的单帧显示

（2）多帧显示

在 MATLAB 中，要同时显示多帧图像阵列，需要调用 montage 函数。其调用格式如下：

```
montage(I)
montage(X,map)%显示索引色图像。
```

下面的例程展示图像的多帧显示。

```
mri = uint8(zeros(128,128,1,27));
for frame=1:27
    [mri(:,:,:,frame),map] = imread('mri.tif', frame);
end
montage(mri, map);
```

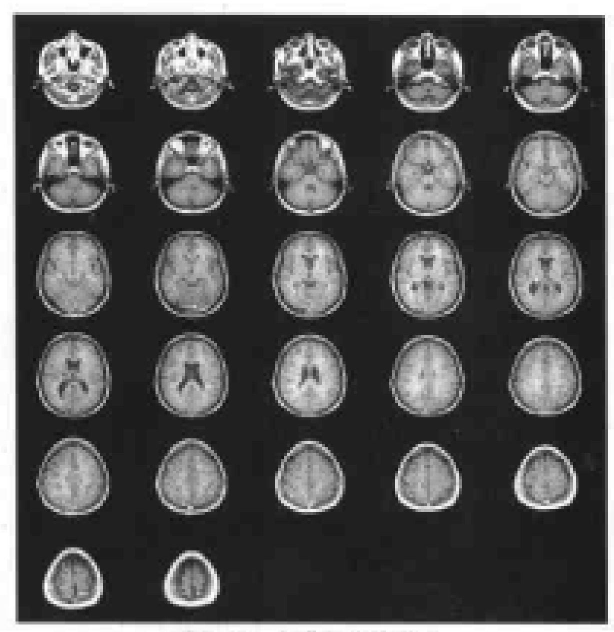


图 2-11　图像的多帧显示

得到如图 2-11 所示的结果。

（3）动画显示

利用 immovie 函数，可以从多帧图像阵列中创建 MATLAB 电影动画。其调用格式如下：

```
mov=immovie(D, map)
mov = immovie(X, map);
```

并可以使用 movie 函数进行播放，调用格式如下：

```
movie(mov);
```

说明：该函数只能用于索引色图像。因此，如果用户希望将其他类型的图像阵列转换为电影动画，必须首先将其类型转换为索引色图像类型。而且函数中所带的参数 D 为一个 $m\times n\times l\times k$ 的 4 维数组，k 代表图像阵列中所包含的帧的数目。

下面的例子中，两幅图像都是灰度图像，所以先利用 gray2ind 函数将它们转换为索引图像，然后还要注意利用已经得到的图像数据矩阵来求数组 D。实现代码如下：

```
mri = uint8(zeros(128,128,1,27));
for frame=1:27
    [mri(:,:,:,frame),map] = imread('mri.tif', frame);
end
mov = immovie(mri,map);
movie(mov);
```

2.3.3　纹理映射

在 MATLAB 中，纹理映射是一种将二维图像映射到三维图形表面的技术。这种技术通过转换颜色数据使二维图像与三维图形表面保持一致。在 MATLAB 中的纹理映射是利用双线性渐变算法来实现图像映射的。

MATLAB 的图像处理工具箱提供了一个专门的函数，即 warp 函数，将图像作为纹理进行映射，使该图像显示在一个特定的三维空间中。下面具体介绍该函数的用法。

warp 函数的调用格式如下：

```
warp(X,map)
warp(I,n)
warp(BW)
warp(RGB)
```

warp(z,…)

warp(x,y,z,…)

h = warp(…)

warp(X,map)、warp(I,n)、warp(BW)和 warp(RGB)分别表示将索引色图像、灰度图像、二值图像和真彩色图像映射到矩形平面区域上显示。由于矩形平面区域本身就是一个二维图形区域，所以调用这四种格式来显示图像与直接调用 imshow 函数的显示结果是一致的。

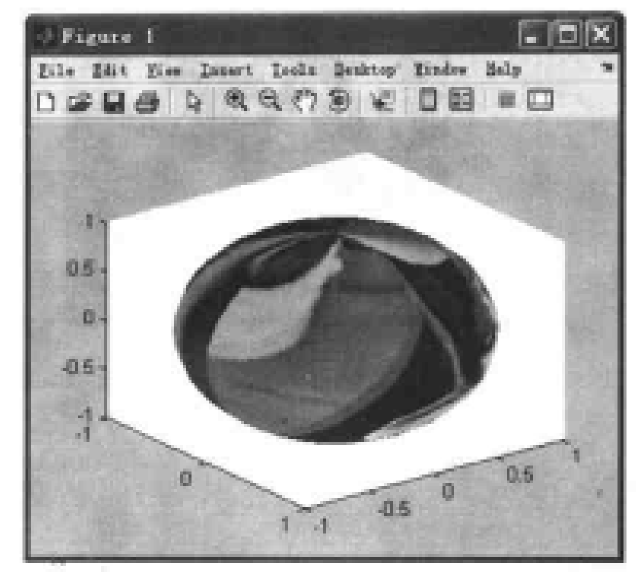


图 2-12 纹理映射效果图

warp(Z,…)表示将图像映射到表面 Z 上。

warp(x, y, z,…)表示将图像映射到由(x, y, z)指定的表面上。

h = warp(…)表示返回纹理映射后的图形句柄。

下面的例程完成将一幅灰度图像映射到球面上。映射后的结果如图 2-12 所示。

```
I= imread('LENA256.bmp');
[x,y,z] = sphere;
warp(x,y,z,I);
```

2.3.4 在一个图形窗口中显示多幅图像

为了便于在多幅图像之间进行比较，我们需要将这些要比较的图像显示在一个图形窗口中。MATLAB 的图像处理工具箱就提供了这样一个函数，即 subimage 函数。下面具体介绍这个函数的格式和用法。

subimage 函数的调用格式如下：

subimage(X,map)

subimage(I)

subimage(BW)

subimage(RGB)

subimage(x,y,…)

h = subimage(…)

subimage(X, map)、subimage(I)、subimage(BW)和 subimage(RGB)分别用于显示索引色图像、灰度图像、二值图像及真彩色图像。

subimage(x,y,…)表示将图像按指定的坐标系(x,y)显示。在具体应用时，主要是设置横轴和纵轴的坐标值范围。

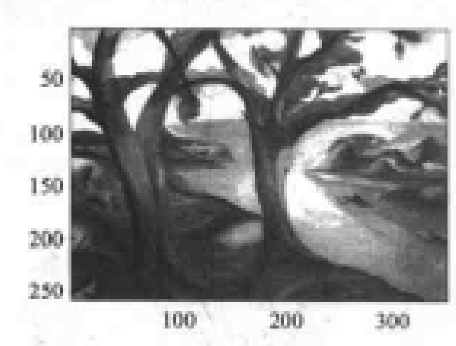
h = subimage(…)表示返回图像对象的句柄。

注意：subimage 函数必须与 subplot 函数一起使用，后者用于指定单个图像的位置。subimage 函数所显示的图像可以是 logical、uint8、uint16 或 double 类型。

下面的例子完成在一个图形窗口中同时显示两幅图像。

```
load trees
[X2, map2] = imread('forest.tif');
subplot(1,2,1), subimage(X, map)
subplot(1,2,2), subimage(X2, map2)
```

程序执行结果如图 2-13 所示。



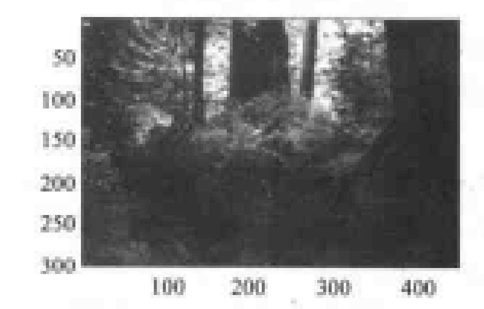


图 2-13 利用函数 subimage 同时显示两幅图像的结果

第 3 章　MATLAB7.0 的图像处理基本操作

在 MATLAB 中，数字图像数据是以矩阵（离散）形式存放的，矩阵的每一个元素值对应着一个像素点的像素值。这样一来，可以对图像数据进行各种应用于数据矩阵上的运算，如代数运算；还可以对图像进行分块操作。当然，由于图像数据的特殊性限制，各种运算结果要符合图像的可视性条件，在这些基本操作中还存在一些不同于一般性矩阵操作的特点，本章将介绍图像数据的基本操作方式及近似规范。

3.1　图像代数操作

图像代数操作是指对图像数据执行矩阵的加、减、乘和除的代数运算。在图像处理中，图像代数操作有很多应用，通常作为一些复杂的图像处理操作的中间环节。在 MATLAB 中，图像数据是采用矩阵格式存储的，然而由于图像数据与一般性的数据矩阵存在处理上的差别，图像代数操作并不能使用简单的矩阵代数操作来执行。为此，MATLAB 的图像处理工具箱提供了一套图像代数操作函数，使得对图像的代数操作变得非常容易。

图像代数操作函数可以处理包括 uint8、uint16 和 double 等各种类型的数值数据，并返回相同类型的结果图像。

3.1.1　图像代数的异常处理

图像数据不同于一般意义上的数据，在执行代数操作得到结果图像的时候，必须注意图像数据的物理意义，保证计算结果的合理性。然而，在执行图像代数操作时，结果经常会出现一些异常情况。常见的异常情况有以下两种。

（1）计算结果溢出。很多图像，如灰度图像、索引色图像、二值图像或有限位真彩色图像，其像素值是有范围限制的，然而在执行两幅或多幅图像的加、减或乘法操作时，计算结果很可能会超出限定的有效范围，比如，两幅 256 色灰度图像在执行减法操作时，很可能会出现像素值为负值的情况，或者执行加法和乘法操作时，像素值超过 255，这都是异常的结果，必须改正。

（2）计算结果类型无效。图像数据有多种存储类型，如 uint8 或 uint16，像素值要求是整数类型，然而在进行除法操作时，往往会得到分数的计算结果，这是因为图像代数操作函数在执行运算时，把图像数据看作是 double 类型。这是另一种异常的图像代数操作结果，也必须加以改正。

MATLAB 中用于普通代数运算的操作符尽管也可以执行加、减、乘和除法运算，但它们对计算结果的有效性不予检查，直接以实数运算的结果进行表示。然而，用于图像的代数操作函数则会自动地对计算结果进行有效性修正。下面分别介绍一下对于以上出现的两种异常结果的修正方式。

异常计算结果的修正遵循两个原则：

（1）超过整数类型有效范围的结果直接截断到限定范围的端点值；

（2）对于分数计算结果采取四舍五入。

例如，如果被处理的图像数据是 uint8 类型的，当计算结果出现以下情况时的修正结果见表 3-1 所示。

表 3-1 像素值取整原则等修正举例

理论计算结果	函数返回值类型	实际输出结果
300	uint8	255
-30	uint8	0
10.5	uint8	11

类似于一般的四则运算，我们可以嵌套使用图像代数函数，即组合多个图像代数函数来完成一系列操作。例如，要计算两幅图像的平均值，用户通常会想到下面的几行代码：

```
I = imread('rice.png');
I2 = imread('cameraman.tif');
K = imdivide(imadd(I,I2), 2);
```

上面的前两条语句是分别读取图像 rice.png 和 cameraman.tif 到变量 I 和 I2，第三条语句是组合使用加法函数 imadd 和除法函数 imdivide，分析起来，这样的代码完全可以完成预定的操作。然而，当我们读进来的图像数据是 uint8 或 uint16 类型时，代数操作函数会自动地按照上面介绍的两个原则对结果进行修正，而且 MATLAB 图像代数函数是每执行一次代数操作就执行一次修正，这样对于第三条语句的嵌套调用则是先对加法运算的结果进行修正再对除法运算的结果进行修正，这就显著地减少了结果图像中包含的大量信息。为了能够得到更好的结果，在嵌套调用图像代数函数时，可以考虑使用函数 imlincomb，这个函数采用线性组合的方式按照 double 类型执行所有的图像代数函数，而且只对最终结果进行数据有效性修正。为此，上面的两幅图像求平均运算，可以采用如下更合适的语句：

```
K = imlincomb(.5,I,.5,I2);
```

3.1.2 相加运算

图像相加运算一般用于对同一场景的多幅图像求平均效果，以便有效地降低具有叠加性质的随机噪声。直接采集的图像品质一般都较好，不需要进行加法运算处理，但是对于那些经过长距离模拟通信方式传送的图像（如卫星图像），这种处理是必不可少的。

在 MATLAB7.0 中，如果要进行两幅图像的加法，或者给一幅图像加上一个常数，可以调用 imadd 函数来实现。imadd 函数将某一输入图像的每一像素值与另一幅图像相应的像素值相加，返回相应的像素值之和作为输出图像的对应像素值。imadd 函数的调用格式可参考图像处理工具箱。下面的程序可将如图 2-6 和图 2-10 所示的两幅图像叠加在一起，叠加效果如图 3-1 所示。



图 3-1 图像相加运算

```
I = imread('Girl.bmp');
J = imread('LENA256.bmp');
K = imadd(I,J,'uint16');
imshow(K,[])
```

给图像的每一个像素加上一个常数可以使图像的整体亮度增加。例如，以下程序实例的处理效果如图 3-2 所示。

```
I = imread('LENA256.bmp');
J=imadd(I, 50);
subplot(1,2,1), imshow(I)
subplot(1,2,2), imshow(J)
```

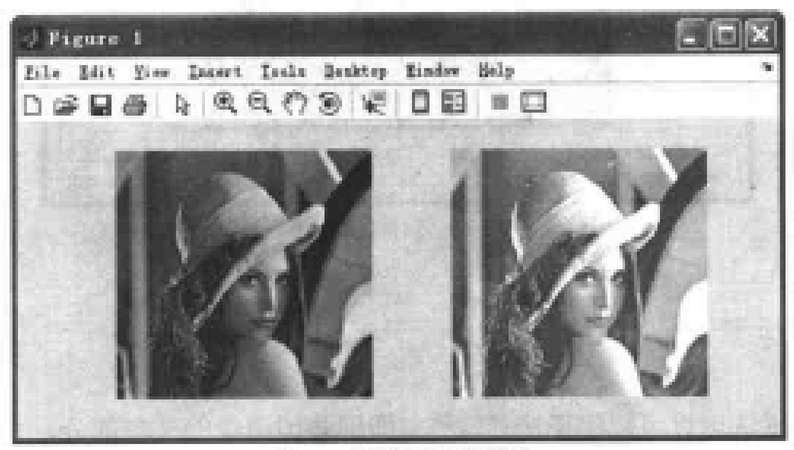


图 3-2　图像亮度整体增强

对两幅图像像素值进行相加操作，其结果很可能超过图像数据类型所支持的最大值，尤其对于 uint8 类型的图像，溢出情况最为常见。当数据值发生溢出时，imadd 函数会将数据截取为数据类型所支持的最大值，这种截取效果称之为饱和处理。为了避免出现饱和现象，在进行加法计算前最好将图像转换为一种数据范围较宽的数据类型。例如，在加法操作前将 uint8 图像转换为 uint16 类型。

3.1.3　减法运算

图像减法也称为差分方法，是一种常用于检测图像变化及运动物体的图像处理方法。图像减法可以作为许多图像处理过程的准备步骤。例如，可以使用图像减法来检测一系列相同场景图像的差异。图像减法与阈值化处理的综合使用通常是建立机器视觉系统最有效的方法之一。当然，在利用图像减法处理图像时，往往需要考虑背景的更新机制，尽量补偿因天气、光照等因素对图像显示效果造成的影响。

在 MATLAB7.0 中，使用 imsubtract 函数可以将一幅图像从另一幅图像中减去，或者从一幅图像中减去一个常数。imsubtract 函数将一幅输入图像的像素值从另一幅输入图像相应的像素值中减去，再将相应的像素值之差作为输出图像相应的像素值。以下的程序代码实例，执行两幅图像相减操作，从而生成如图 3-3 所示的图像。

```
I = imread('LENA256.bmp');
J=imread('Girl.bmp');
```

```
Iq = imsubtract(I,J);
imview(Iq)
```

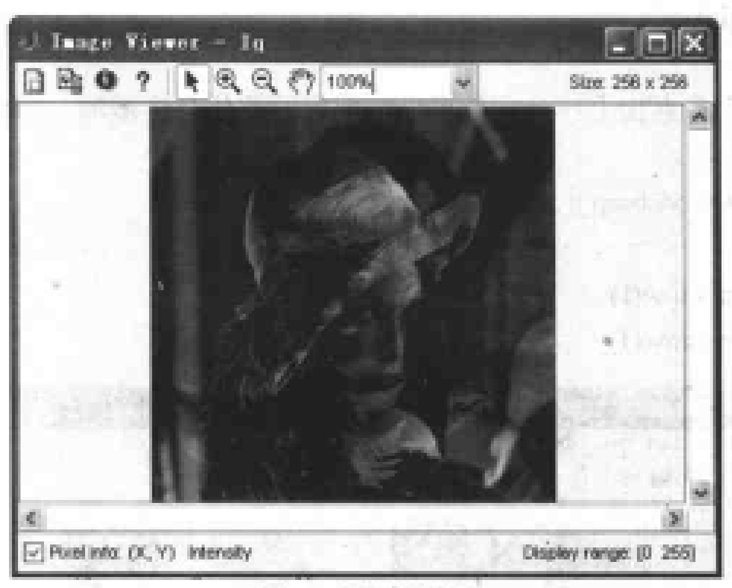


图 3-3　图像减法运算

3.1.4　乘法运算

两幅图像进行乘法运算可以实现掩模操作，即屏蔽掉图像的某些部分。一幅图像乘以一个常数通常被称为缩放，这是一种常见的图像处理操作。如果使用的缩放因数大于 1，那么将增强图像的亮度，如果因数小于 1 则会使图像变暗。缩放操作通常将产生比简单添加像素偏移量自然得多的明暗效果。这是因为该操作能够更好地维持图像的相关对比度。此外，由于时域的卷积或相关运算与频域的乘积运算对应，因此乘法运算有时也成为一种技巧来实现卷积或相关处理。

在 MATLAB7.0 中，可以使用 immultiply 函数实现两幅图像的乘法或一幅图像的亮度缩放。immultiply 函数将两幅图像相应的像素值进行元素对元素的乘法操作，即图像矩阵的点乘运算，并将乘法的运算结果作为输出图像相应的像素值。例如，以下程序实例将使用给定的缩放因数对如图 3-4 左图所示的图像进行亮度缩放，从而得到如图 3-4 右图所示的较为暗淡的图像。

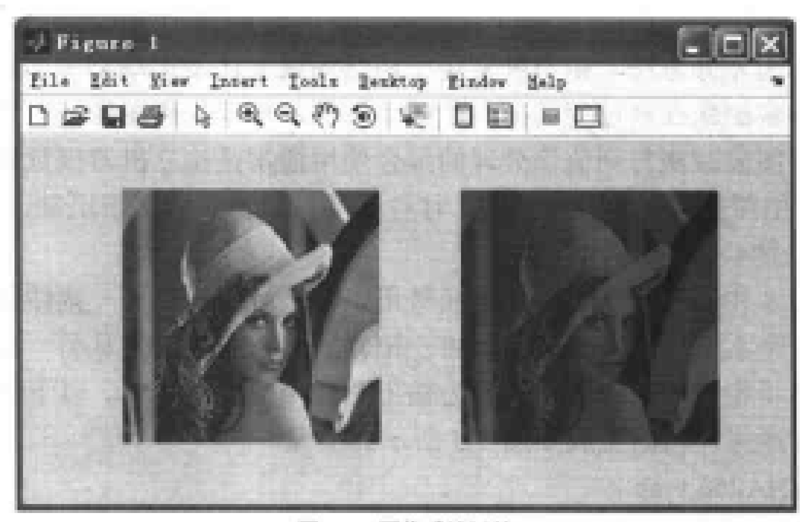


图 3-4　图像乘法运算

```
I = imread('LENA256.bmp');
J = immultiply(I,0.5);
subplot(1,2,1), imshow(I)
subplot(1,2,2), imshow(J)
```

uint8 类型的图像在进行乘法操作时，一般都会发生溢出现象。immultiply 函数将溢出的数据截取为数据类型的最大值。为了避免产生溢出现象，可以在执行乘法操作之前将 uint8 类型的图像转换为一种数据范围较大的图像类型。

3.1.5 除法运算

除法运算可用于校正成像设备的非线性影响，这在特殊形态的图像（如断层扫描等医学图像）处理中经常用到。图像除法也可以用来检测两幅图像间的区别，但是除法操作给出的是相应像素值的变化比率，而不是每个像素的绝对差异，因而图像除法操作也称为比率变换。

在 MATLAB7.0 中，可以使用 imdivide 函数进行两幅图像的除法或一幅图像的亮度缩放。imdivide 函数对两幅输入图像的所有相应像素执行元素对元素的除法操作（点除），并将得到的结果作为输出图像的相应像素值。以下程序代码实例将如图 3-5 左图所示的图像进行除法操作，得到如图 3-5 右图所示的效果。

```
I = imread('LENA256.bmp');
J = imdivide(I,0.5);
subplot(1,2,1), imshow(I)
subplot(1,2,2), imshow(J)
```

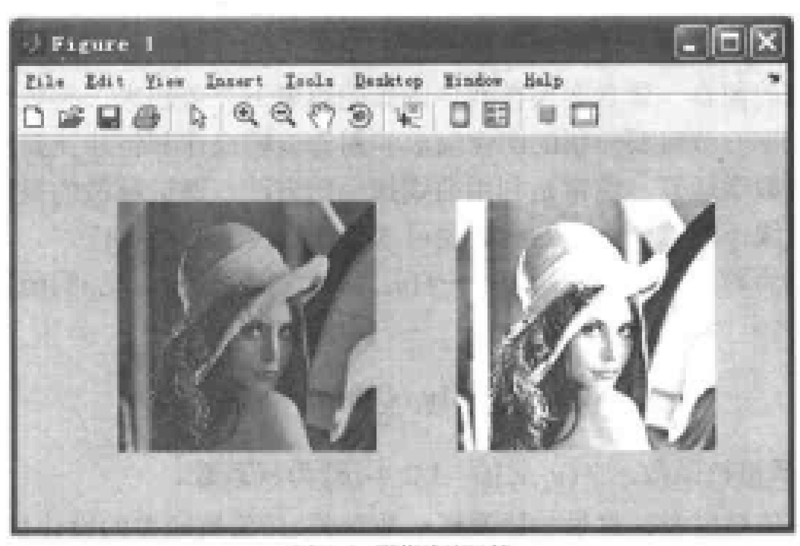


图 3-5 图像除法运算

3.2 图像的空间域变换操作

简单地说，图像的空间域变换操作就是为了达到某种视觉效果，变换输入图像的像素位置，通过把输入图像的像素位置映射到一个新的位置以达到改变原图像显示效果的目的。其

实，也就是变换图像的坐标系统。图像的空间域变换操作包括以下几个方面：

- 图像插值（Interpolation）
- 图像缩放（Resizing）
- 图像旋转（Rotation）
- 图像剪切（Cropping）
- 一般变换操作（Affine，Projective，Box，Custom，etc）

在处理图像的过程中，有时需要对图像的大小和几何关系进行调整，比如对图像进行缩放及旋转，这时图像中每个像素的值都要发生变化。数字图像的坐标是整数，经过这些变换之后的坐标不一定是整数，因此要对变换之后的整数坐标位置的像素值进行估计。MATLAB提供了一些函数实现这些功能。

3.2.1 图像插值

图像空间域变换操作可以认为是在输入图像和输出图像之间进行像素—像素变换，然而由于我们处理的是数字图像，其重要特点就是图像的横纵坐标值是离散的，这就使得执行空间域变换操作（如缩放、旋转等）后的输出图像中存在像素点无法找到其在输入图像中的对应点，而输入图像的像素点经过空间域变换后也可能落在输出图像中的无效位置上，因而输出图像中将会出现空白点，图像失去可视性，如图3-6所示。

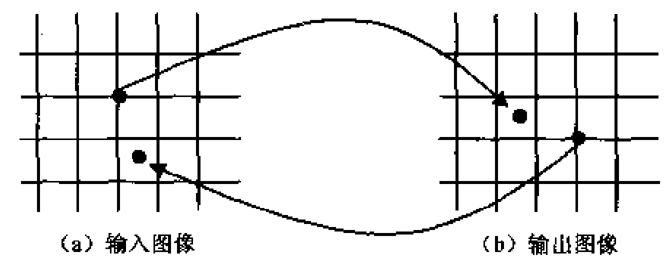


图 3-6 图像插值问题的产生

为了弥补这一显示缺陷，需要引入图像插值操作，估计像素点之间位置的像素值，将输入图像和输出图像的像素—像素变换在数字图像的约束下完善起来，有效地填充图像中可能出现的空白点。在下一节将要介绍的图像缩放和图像旋转操作都将涉及到图像插值运算。

插值是常用的数学运算，通常是利用曲线拟合的方法，通过离散的采样点建立一个连续函数来逼近真实曲线，用这个重建的函数便可求出任意位置的函数值。

设已知函数值为$f(u_1, v_1)$、$f(u_2, v_2)$……$f(u_n, v_n)$，则未知点(u_0, v_0)的函数值通过插值可以表示为：

$$f(u_0,v_0)=\sum_{i=1}^{n}f(u_i,v_i)h(u_i-u_0,v_i-v_0) \tag{3.1}$$

其中$h(\cdot,\cdot)$为插值核函数，$f(u_i,v_i)(i=1,2,\cdots,n)$为权系数。

插值算法的数值精度及计算量与插值核函数有关，插值核函数的设计是插值算法的核心。MATLAB图像处理工具箱提供了3种插值方法：最近邻插值（Nearest Neighbor Interpolation）、双线性插值（Bilinear Interpolation）和双三次插值（Bicubic Interpolation）。

使用插值操作的空间域变换操作，如图像缩放、图像旋转及其他一些一般性变换（仿射变换、投影变换或用户自定义变换），一般需要在其相应操作中指明所使用的插值方法作为函数参数。如未指定，则系统默认执行最近邻插值方式。另外，针对插值操作，还需要注意以下两个方面：

（1）对于RGB图像而言，插值操作将对其R、G、B 3个分量分别进行。在后面的介绍中，读者将会看到MATLAB提供的变换操作函数可以对其不同分量指定不同的插值方式。

（2）对于二值图像，在进行双线性插值或双三次插值时，计算得到的像素值可能不再是只有 0 和 1 两个值，结果依赖于输入图像数据的存储类型。如果输入图像的存储类型是 double 类型，则输出图像将会包括 0 和 1 以外的值；而对于 uint8 类型的输入图像，由于在插值过程中进行了四舍五入的操作，得到的输出图像仍为 uint8 类型的二值图像。但是，当采用最近邻插值时，无论输入图像数据是何种存储类型，得到的始终都是二值图像。

下面简单介绍 3 种插值方法。

1. 最近邻插值

最近邻插值是最简单的插值算法，在这种算法中，输出图像中每一个像素点的值就是与该点在输入图像中变换位置最临近采样点的值。该算法的数学表示为：

$$f(u_0,v_0)=f(u_k,v_k), \text{ 如果}\begin{cases}u_k-1/2<u_0<u_k+1/2\\ v_k-1/2<v_0<v_k+1/2\end{cases} \tag{3.2}$$

最近邻插值方法的运算量非常小，是图像空间域变换操作函数默认使用的插值方法。对于索引色图像来讲，它是唯一可用的插值方法。不过，最近邻插值法的核函数频域特性不是很好，从它的傅立叶频谱上可以看出，它与理想低通滤波器的性质相差较大。当用这种方法实现大倍数放大处理时，在图像中会明显看出块状效应。

2. 双线性插值

双线性插值法的输出像素值是它在输入图像中 2×2 邻域采样点的平均值，它根据某输出像素点在输入图像变换位置周围 4 个像素的灰度值在水平和垂直两个方向上对其插值，如图 3-7 所示。

设 (u_0,v_0) 是要插值点的坐标，则双线性插值的方法为：

第一步，从 $f(u_k,v_k)$ 及 $f(u_k+1,v_k)$ 求 $f(u_0,v_k)$；

第二步，从 $f(u_k,v_k+1)$ 及 $f(u_k+1,v_k+1)$ 求 $f(u_0,v_k+1)$。

把按照上式计算出来的值赋予图像的几何变换对应于 (u_0,v_0) 处的像素，即可实现双线性插值。

3. 双三次插值

双三次插值的插值核为三次函数，其插值邻域的大小为 4×4。它的插值效果相比前两种方法要好，但相应的计算量也较大。为了计算 (u_0,v_0) 点的像素值，采用双三次插值法的示意图如图 3-8 所示。

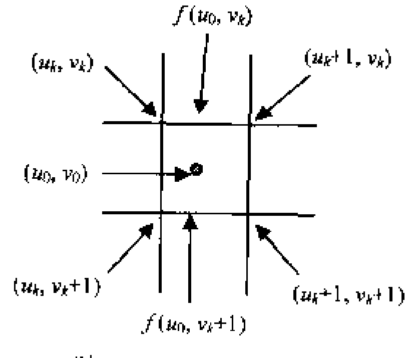


图 3-7 双线性插值示意图

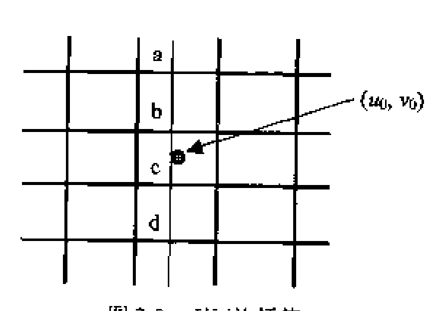


图 3-8 双三次插值

第一步，在四条水平直线上分别用三次多项式插值计算点 a、b、c、d 处的灰度值；

第二步，对 a、b、c、d4 点在垂直方向上再做 3 次多项式内插，得到 (u_0, v_0) 点的像素值。

3.2.2 图像缩放

MATLAB 图像处理工具箱中的函数 imresize 可以对图像进行缩放操作，同时指定以上所介绍的插值方法作为其函数参数，如不指定，则默认为是最近邻插值法。

imresize 函数的语法格式为：

```
B = imresize(A, m)
B = imresize(A,m, method)
B = imresize(A,[mrows ncols], method)
B = imresize(…,method, n)
B = imresize(…,method, h)
```

这些语句的功能就是按照指定的放大倍数和插值方法等参数将图像 A 缩放成图像 B。其中，参数 m 表示放大倍数，$m>1$ 表示图像放大，$m<1$ 表示图像缩小；参数 method 表示选用的插值方法，其可选值为 nearest（最近邻插值）、bilinear（双线性插值）和 bicubic（双三次插值）。

B=imresize(A,[mrows ncols], method)语句通过[mrows ncols]指定了输出图像的大小为 mrows×ncols，从而达到了对原图像 A 进行指定大小缩放的目的。

B=imresize(…,method, n)语句在 imresize 函数中引入了一个表征滤波器大小的整数作为参数。在使用双线性插值和双三次插值法缩小图像时，为消除引入的高频成分，imresize 在插值之前使用一个低通滤波器来减弱混叠现象，n 就是这个低通滤波器的大小，表示 $n \times n$ 的窗口，默认值为 11×11。如果 n 取值为 0，则表示忽略滤波步骤。

B=imresize(…,method, h)语句允许用户自行指定滤波器，其中 h 表示一个二维 FIR 滤波器，例如由函数 ftrans2、fwind1、fwind2 或 fsamp2 生成的滤波器结构。

如图 3-9 所示，图（a）为原图像，图（b）、图（c）和图（d）分别表示采用最近邻插值法、双线性插值法和双三次插值法得到的图像放大结果。其实现代码如下：



（a）原图像



（b）最近邻插值法



（c）双线性插值法



（d）双三次插值法

图 3-9 采用不同插值方法的图像缩放

```
J=imread('lena.bmp');
X1=imresize(J,2);
X2=imresize(J,2,'bilinear');
X3=imresize(J,2,'bicubic');
figure,imshow(J)
figure,imshow(X1)
```

figure,imshow(X2)

figure,imshow(X3)

从以上 3 种插值方法对图像放大的结果可以看出，采用最近邻插值法的放大图像中明显可以看出块状效应，但对于质量要求不高的情况下，效果还是还可以接受的，双线性插值法和双三次插值法的结果则没有块状效应，但双线性插值法有些模糊，双三次插值法效果最好。

3.2.3 图像旋转

在对数字图像进行旋转的时候，各像素的坐标将会发生变化，使得旋转之后不能正好落在整数坐标处，需要进行插值。在工具箱中的函数 imrotate 可用上述 3 种方法对图像进行插值旋转，默认的插值方法也是最近邻插值法。

imrotate 的语法格式为：

B = imrotate(A, angle)

B = imrotate(A, angle, method)

B = imrotate(A, angle, method, bbox)

函数 imrotate 对图像 A 进行旋转得到图像 B，参数 angle 用于指定图像按逆时针方向旋转的角度，参数 method 用于指定插值的方法，可选的值为 nearest（最近邻插值）、bilinear（双线形插值)及 bicubic（双三次插值），默认值为 nearest。一般来说，旋转后的图像会比原图大，超出原图像的部分值为 0。用户也可以指定 crop 参数对旋转后的图像进行剪切（取图像的中间部分），使返回的图像与原图大小相同。

下面的例子可将图像插值旋转 45°，效果如图 3-10 所示。

```
I=imread('lena1.bmp');
J=imrotate(I,45,'bilinear');
figure,imshow(I)
figure,imshow(J)
```




图 3-10 采用双线性插值法的图像旋转（45°）

3.2.4 图像剪切

当只需要处理图像中的一部分时，或者需要将某一部分取出，这样就要对图像进行剪切。图像处理工具箱提供函数 imcrop 用于剪切图像中的一个矩形子图，用户可以通过参数指定这

个矩形顶点的坐标，也可以用鼠标指针选取这个矩形。

imcrop 函数的语法格式为：

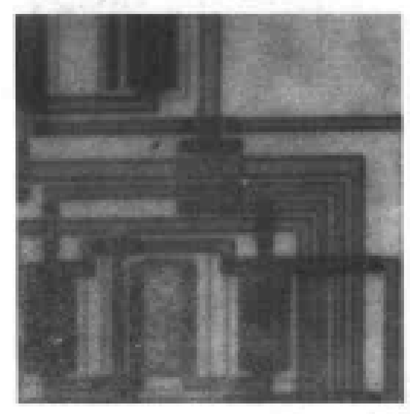
```
I2 = imcrop(I)
X2 = imcrop(X,map)
RGB2 = imcrop(RGB)
I2 = imcrop(I,rect)
X2 = imcrop(X,map,rect)
RGB2 = imcrop(RGB,rect)
[…] = imcrop(x,y,…)
[A,rect] = imcrop(…)
[x,y,A,rect] = imcrop(…)
```

其中 I2=imcrop(I)、X2=imcrop(X, map)和 RGB2=imcrop(RGB)为交互式地对灰度图像、索引色图像和真彩色图像进行剪切。I2=imcrop(I,rect)、X2=imcrop(X,map,rect)和 RGB2=imcrop(RGB,rect)按指定的矩形框 rect 剪切图像，rect 是一个四元向量[xmin ymin width height]，分别表示矩形的左上角的坐标、宽度和高度。[…] = imcrop(x,y,…)在指定坐标系(x,y)中剪切图像。[A,rect]=imcrop(…)和[x,y,A,rect]=imcrop(…)在用户手动选取剪切图像的同时返回剪切框的参数 rect。

下面的例子将从一幅图像中剪切一块子图，坐标为(75,68)~(205,180)，结果如图 3-11 所示。

```
I = imread('circuit.tif');
I2 = imcrop(I,[75 68 130 112]);
imview(I), imview(I2)
```

因为 rect 是借助于空间坐标指定的，rect 的宽度（width）和高度（height）并不总是恰好对应着输出图像的大小。本例中，指定 rect 左上角为像点(75,68)的中心，宽和高分别为 130 和 112，即矩形的右下角在原图像中的像点(205,180)的中心，因此得到的输出图像大小为 131×113。这是因为输出图像包含了被指定矩形完全或部分包围的全部像点，等于是在指定大小的基础上多了几条边界像点。



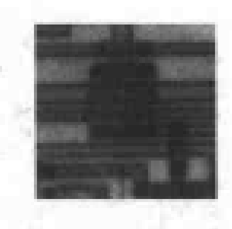


图 3-11　图像剪切

3.2.5　高级空间域变换

以上介绍的缩放、旋转和剪切是比较常见的空间域变换操作。要执行通用的二维空间变

换，MATLAB 提供了一些高级空间域变换函数如下：

maketform

fliptform

tformfwd

tforminv

findbounds

makeresampler

tformarray

imtransform

这其中最常用的变换函数是 imtransform，该函数接受两个基本的参数，即待变换图像和变换结构。得到变换结构的方式有两种，即调用函数 maketform 和函数 cp2tform。这两个函数又通常包括一个共同的参数，即变换类型。MATLAB 支持的变换类型见表 3-2 所示。

表 3-2　　MATLAB 支持的变换类型

变换类型	描　述
'affine'	这种变换包含平移（Translation）、旋转（Rotation）、按比例缩放（Scaling）、拉伸（Stretching）以及减切（Shearing）等。经过变换，直线仍为直线，平行线仍然相互平行，但矩形可能会变成平行四边形。
'box'	仿射（Affine）变换的一个特例，变换时每一维数据独立地进行移动和按比例缩放。
'composite'	两种或两种以上变换的合成。
'custom'	用户自定义的变换，提供用于函数 imtransform 调用的正向变换函数或反向变换函数。
'projective'	投影变换，经过这类变换，直线仍然保持为直线，但平行线会聚到消失点。

下面简要介绍这几个工具箱函数的使用。

（1）maketform

该函数用于生成一个多维空间变换结构，为函数 tformfwd、tforminv、fliptform、imtransform 或 tformarray 提供变换参数。其语法格式为：

T = maketform(transformtype,⋯)

其中，transformtype 是指变换类型，其取值为'affine'、'box'、'composite'、'custom'或'projective'中的一种，对于不同的变换类型，该函数的语法结构也不同。

- Affine

语法格式有两种：

T = maketform('affine',A)

该语句为一个 N 维的仿射变换 A 生成一个变换结构 T。A 是一个非奇异的(N+1)×(N+1)或者(N+1)×N 的实矩阵。如果 A 的大小是(N+1)×(N+1)，则 A 的最后一列为[zeros(N,1);1]，即只有最后一个元素为 1 其余元素为 0 的 N+1 维矢量，如果 A 的大小是(N+1)×N，则 A 被自动地扩充为(N+1)×(N+1)的矩阵，最后一列为[zeros(N,1);1]。矩阵 A 定义了一个正向变换 tformfwd(U,T)，其中 U 是一个 N 维的行向量，返回值为 N 维行向量 X，X = U * A(1:N,1:N) + A(N+1,1:N)，T 既有正向变换又有反向变换。

T = maketform('affine',U,X)

该语句建立了一个二维仿射变换结构，该变换映射 U 的每一行到 X 的对应行。U 和 X 都是 3×2 的矩阵，定义了输入和输出图像中三角形的顶点，要求这些顶点不能是共线的。

- Projective

T = maketform('projective',A)

该语句建立一个二维投影变换结构 T。A 是一个非奇异的(N+1)×(N+1)的实矩阵，定义一个正向变换，tformfwd(U,T),其中 U 是一个 N 维的行向量，该变换函数返回一个 N 维的行向量 X，X = W(1:N)/W(N+1)，其中 W = [U 1] * A。T 既有正向变换又有反向变换。

T = maketform('projective',U,X)

该语句建立了一个二维投影变换结构 T，该变换映射 U 的每一行到 X 的对应行。U 和 X 参数都是 4×2 的矩阵，定义了输入和输出图像中的四边形的顶点，要求 4 个顶点中任意 3 个都不共线。

- Custom

T = maketform('custom', NDIMS_IN, NDIMS_OUT, FORWARD_FCN, INVERSE_FCN, TDATA)

该语句基于用户提供的函数句柄和参数建立了一个用户自定义的变换结构 T。NDIMS_IN 和 NDIMS_OUT 分别表示输入和输出数据的维数。FORWARD_FCN 和 INVERSE_FCN 正函数和反函数句柄，而且这些函数必须支持以下语法：

Forward function: X = FORWARD_FCN(U,T)

Inverse function: U = INVERSE_FCN(X,T)

这些函数的具体意义可参考帮助文件的相关内容。

- Box

T = maketform('box',tsize,LOW,HIGH)或 T = maketform ('box', INBOUNDS, OUTBOUNDS)

该语句建立了一个 N 维盒式变换结构 T。参数 tsize 和 LOW、HIGH 都是 N 个元素的矢量，其中 tsize 的元素都为正整数。该变换实现将输入数据中的一个 box 结构映射到输出数据中的一个 box 结构，这个 box 结构可以由 ones(1,N)和 tsize 的对角点或者由顶点 INBOUNDS(1,:) 和 INBOUND(2,:) 定义。这种变换的典型应用是配准图像的行列坐标到世界坐标系的变换。

- Composite

T = maketform('composite',T1,T2,…,TL)或 T = maketform('composite', [T1 T2…TL])

该语句建立了一个合成变换结构 T，其正函数和反函数是另一些正、反函数 T1, T2,…,TL 等的合成。

例如，当 L = 3，那么 tformfwd(U,T)就等价于 tformfwd(tformfwd (tformfwd (U,T3) ,T2), T1)。合成元素 T1, T2,…, TL 在输入和输出数据的维数上必须一致。

下面是一个生成和应用仿射变换的例子：

```
T = maketform('affine',[.5 0 0; .5 2 0; 0 0 1]);
tformfwd([10 20],T)
I = imread('cameraman.tif');
I2 = imtransform(I,T);
imshow(I2)
```

（2）tformfwd

该函数用于实现正向空间变换，其语法格式如下：

```
[X,Y] = tformfwd(T,U,V)
[X1,X2,X3,…] = tformfwd(T,U1,U2,U3,…)
X = tformfwd(T,U)
```

[X1,X2,X3,⋯] = tformfwd(T,U)

X = tformfwd(T,U1,U2,U3,⋯)

[X,Y] = tformfwd(T,U,V)实现 2D-2D 的空间变换 T，映射点 [U(k) V(k)]到点[X(k) Y(k)]。T 可以由函数 maketform、fliptform 或 cp2tform 得到。T.ndims_in 和 T.ndims_out 必须等于 2。U 和 V 通常可以是多维矢量，但维数必须相同。X 和 Y 与 U 和 V 大小相同。

[X1,X2,X3,⋯] = tformfwd(T,U1,U2,U3,⋯)应用变换结构 T，映射点[U1(k) U2(k)⋯UNDIMS_IN(k)]到点[X1(k) X2(k)⋯XNDIMS_OUT(k)]。

X = tformfwd(T,U)应用变换结构 T 作用在 U 的每一行，其中 U 是一个 M×N DIMS_IN 的矩阵。该变换映射点 U(k,:)到点 X(k,:)。X 是一个 M×N DIMS_OUT 的矩阵。

[X1,X2,X3,⋯] = tformfwd(T,U)映射一个(N+1)维的数组到 NDIMS_OUT 等于 N 的数组。

X = tformfwd(T,U1,U2,U3,⋯)映射一个 NDIMS_IN 维的数组到一个(N+1)维的数组。

注意： X = tformfwd(U,T)是一个旧格式，为了保持向下兼容，该格式仍然有效。

下面的例子给出了函数 tformfwd 的应用。生成一个仿射变换结构，映射一个顶点为(0,0), (6,3), (−2,5)的三角形到顶点为(−1, −1), (0, −10), (4,4)的三角形。

```
u = [0    6   -2]';
v = [0    3    5]';
x = [-1    0    4]';
y = [-1   -10   4]';
tform = maketform('affine',[u v],[x y]);
```

应用函数 tformfwd，下面的语句将得到结果[x, y]。

```
[xm, ym] = tformfwd(tform, u, v)
```

（3）tforminv

反向空间变换，语法结构与 tformfwd 类似，不再赘述。下面举一个例子说明该函数的应用。与上一个例子相同，首先生成一个仿射变换，映射一个顶点为(0,0), (6,3), (−2,5)的三角形到顶点为(−1, −1), (0, −10), (4,4)的三角形。

```
u = [0    6   -2]';
v = [0    3    5]';
x = [-1    0    4]';
y = [-1   -10   4]';
tform = maketform('affine',[u v],[x y]);
```

应用 tforminv，下面的语句将得到结果 [u, v]。

```
[um, vm] = tforminv(tform, x, y)
```

（4）fliptform

该函数生成一个新的空间变换结构，颠倒原变换结构中输入和输出的角色。语法结构如下：

```
TFLIP = fliptform(T)
```

举例：

```
T = maketform('affine', [.5 0 0; .5 2 0; 0 0 1]);
T2 = fliptform(T)
```

那么，下面的两条语句是等效的。

```
x = tformfwd([-3 7],T)
```

```
x = tforminv([-3 7],T2)
```

（5）findbounds

该函数用于找到空间变换的输出边界。语法格式为：

```
outbounds = findbounds(TFORM,inbounds)
```

其中，TFORM 是一个空间变换结构，inbounds 是一个输入边界。outbounds 与 inbounds 有相同的形式。不过，outbounds 表示一个矩形边界，要求该矩形能够完全包含输入边界所代表矩形的变换，由于只是一个估计，因此，outbounds 可能不会完全地包含被变换的输入矩形。

举一个例子说明该函数的用法：

```
inbounds = [0 0; 1 1]
tform = maketform('affine',[2 0 0; .5 3 0; 0 0 1])
outbounds = findbounds(tform, inbounds)
```

（6）makeresampler

该函数用于生成重采样（resampler）结构，其语法结构如下：

```
R = makeresampler(interpolant,padmethod)
```

参数 interpolant 用于指定重采样所用的插值核，它的取值通常有以下几种：'nearest'、'linear'和'cubic'，即最近邻插值、线性插值和三次插值，另外，也可以采用用户自定义的类型值。参数 padmethod 控制重采样函数如何为接近图像边缘或在图像边缘之外的输出像素进行插值或指派有效值。padmethod 可能的取值有'bound'、'circular'、'fill'、'replicate'和'symmetric'。

该函数的应用举例如下：

```
A = imread('moon.tif');
resamp = makeresampler({'nearest','cubic'},'fill');
stretch = maketform('affine',[1 0; 0 1.3; 0 0]);
B = imtransform(A,stretch,resamp);
```

（7）tformarray

该函数用于高维的空间变换，语法格式如下：

```
B = tformarray(A,T,R,TDIMS_A,TDIMS_B,TSIZE_B,TMAP_B,F)
```

其中各个参数的含义如表 3-3 所示。

表 3-3　　函数 **tformarray** 的参数含义

参　数	含　义
A	输入数组或图像
T	空间变换结构，通常是由 maketform 得到的
R	空间变换结构，通常是由 makeresampler 得到的
TDIMS_A	匹配输入变换维数的行矢量
TDIMS_B	匹配输出变换维数的行矢量
TSIZE_B	变换维数表示的输出数组大小
TMAP_B	输出空间点的位置数组
F	填充值数组

（8）imtransform

该函数用于对图像进行二维空间变换，其语法格式为：

```
B = imtransform(A,TFORM)
```

```
B = imtransform(A,TFORM,INTERP)
[B,XDATA,YDATA] = imtransform(…)
[B,XDATA,YDATA] = imtransform(…,param1,val1,param2,val2,…)
```

B=imtransform(A,TFORM)表示根据2D空间变换结构TFORM变换图像A得到输出图像B。如果A的维数大于2，如RGB图像，那么通用的2D变换被自动应用到所有的2D平面中。

B = imtransform(A,TFORM,INTERP)指定了插值的形式，INTERP可以取'bicubic'、'bilinear'和'nearest'，默认值为'bilinear'。

另外，INTERP也可以是一个由makeresampler得到的重采样结构RESAMPLER。

[B,XDATA,YDATA] = imtransform(…)返回输出坐标 x–y 空间中输出图像B的位置。XDATA and YDATA是包含两个元素的矢量。XDATA的元素指定了B的第一和最后一列的x坐标，YDATA的元素指定了B的第一和最后一行的y坐标。

[B,XDATA,YDATA] = imtransform(…,param1,val1,param2,val2,…)指定了空间变换不同方面的控制参数。

下面是一个投影变换的例子，该投影变换映射一个正方形到一个任意四边形。在这个例子中，根据预先设定的坐标系统，正方形的四个顶点为(0, 0), (1, 0), (1, 1), (0, 1)，映射得到的四边形的四个顶点为(−4, 2), (−8, 3), (−3, −5), (6, 3)。用灰色填充正方形，并采用'bicubic'插值方法。实现代码如下：

```
I = imread('cameraman.tif');
udata = [0 1];   vdata = [0 1];   %输入坐标系统
tform = maketform('projective',[ 0 0;   1   0;   1   1; 0 1],…
                                [-4 2; -8 -3; -3 -5; 6 3]);
[B,xdata,ydata] = imtransform(I, tform, 'bicubic',…
                              'udata', udata,…
                              'vdata', vdata,…
                              'size', size(I),…
                              'fill', 128);
subplot(1,2,1), imshow(udata,vdata,I), axis on
subplot(1,2,2), imshow(xdata,ydata,B), axis on
```

3.3 图像的邻域和块操作

在很多图像处理过程中，对图像分块操作而不是同时处理整幅图像的方法是非常通用而且有效的，尤其是在后面章节中将要介绍的图像滤波和图像形态学操作中有很重要的应用。相比全图像操作，图像分块操作至少有以下3个优点：

- 节省运算时占用的存储空间；
- 降低计算的复杂性，提高处理速度；
- 充分考虑图像的局部特性。

本节将会结合图像块操作的类型，介绍几个通用的图像块操作函数。这些块操作函数通常需要指定图像块的大小和对图像块进行特定处理的功能函数。图像块操作的类型有两种：

- 非重叠块操作（distinct block）
- 滑动邻域操作（sliding neighborhood）

这两种操作都是基于一个图像块的整体操作，按照指定的图像处理功能函数得到仅涉及该图像块的计算结果，而且，在对原图像进行块操作时，都可以根据需要指定不同大小的矩形块。然而，不同的是，非重叠块操作对原图像的划分不存在重叠区域，而且在对某一图像块进行处理后，得到的是对应输出图像中同样大小的图像块；滑动邻域操作是基于像素级的操作模式，以原图像某像素点为中心，以指定大小的图像块作为该像素点的邻域，对该像素点的整个邻域进行处理，每次处理后仅得到输出图像中对应像素点的值，而不是一个图像块的计算结果，而且，接下来不改变邻域的大小，仅仅移动邻域的中心像素位置，从而计算出所有的输出图像像素值，在这一过程中相邻图像邻域很明显有重叠区域。

另外，针对图像块操作的两种类型，MATLAB 工具箱提供了一个快速处理的实现方法，即将得到的非重叠图像块或者滑动邻域按列进行处理，得到一个存储所有变换列的临时矩阵，利用 MATLAB 强大的向量处理功能，达到提高整体处理速度的目的。在本节第三部分，将会对此进行介绍。

3.3.1 非重叠图像块操作

非重叠图像块的定义将图像的数据矩阵划分为同样大小的矩形区域，不同的图像块在图像上面排列，相互之间没有重叠。它的排列顺序是从左上角开始。如果图像不能恰好被划分，则在图像的右、下部对其补零。如图 3-12 所示，一个 15×30 的图像按照 4×8 的图像块进行划分，结果在图像的右边补了两列“0”，在图像的下边补了一行“0”。

另外，对于一些特别的图像处理操作，可能还需要进行一些特别的图像块划分，即包含有“重叠（Overlap）”定义区域的非重叠（Distinct）图像块划分方式，如图 3-13 所示。

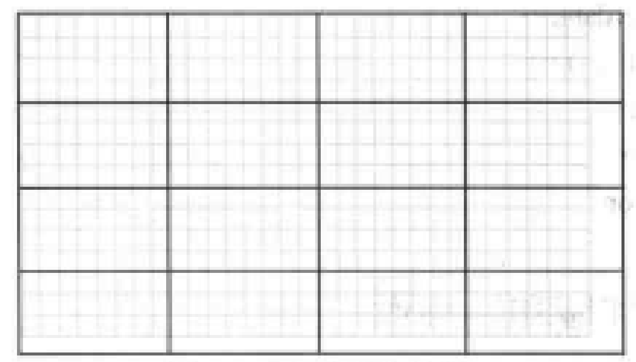


图 3-12　非重叠图像块示意图

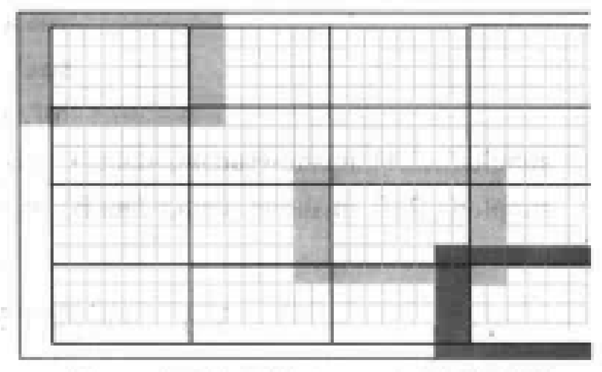


图 3-13　包含有“重叠（Overlap）”定义区域的非重叠（Distinct）图像块示意图

图中阴影区域表示对图像块进行补“0”操作。按照前面介绍的，“非重叠（Distinct）”的含义是针对原图像而言，不同的图像块中不会重叠包含原图像中相同的像素点，而“重叠（Overlap）”的意义是在对划分的每个图像块（包括左上角第一个图像块在内）经过补“0”后，不同图像块之间含有重叠的“0”值像素区。

MATLAB 提供了一个通用的非重叠图像块操作函数 blkproc，其语法格式为：

B = blkproc(A,[m n],fun)

B = blkproc(A,[m n],fun,P1,P2,⋯)

B = blkproc(A,[m n],[mborder nborder],fun,⋯)

B = blkproc(A,'indexed',⋯)

B = blkproc(A,[m n],fun)其中[m n]表示按 m×n 的图像块划分对图像 A 做运算，fun 为运算函数，其形式为 y=fun(x)，x 表示被操作的图像块。

B=blkproc(A,[m n],fun,Pl,P2,⋯)指定 fun 中除 x 以外的其他参数 P1、P2、⋯⋯

B=blkproc[A,[m n],[mborder nborder],fun，⋯]指定图像块的扩展边界 mborder 和 nborder，使得实际操作的图像块大小为(m+2*mborder)*(n+2*nborder)。如图 3-13 所示的包含有“重叠（Overlap）”定义区域的非重叠（Distinct）图像块划分方式，阴影部分即为扩展边界大小，在该图中，扩展边界[mborder nborder]为[1 2]。

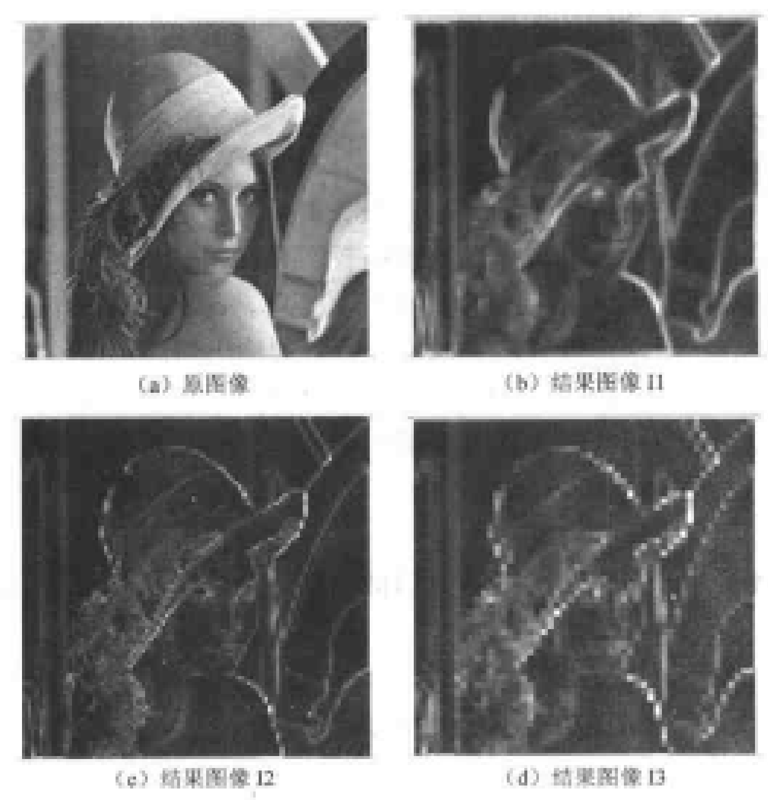
B = blkproc(A,'indexed',⋯)表示被处理图像 A 为索引色图像。

下面的例子为计算图像的局部标准差，实现代码如下：

```
I = imread('LENA256.bmp');
fun = inline('std2(x)*ones(size(x))');
I1 = blkproc(I,[ 3 3 ],[2 2],fun);
I2 = blkproc(I,[ 3 3 ],fun);
I3 = blkproc(I,[ 5 5 ],fun);
imview(I), imview(I1,[]) , imview(I2,[]) , imview(I3,[])
```

说明：图像块的标准差计算公式为 std2(x)，结果为标量。但是为了使运算之后的图像与原图像大小相同，因此乘以一个与图像块大小相同的方阵 ones(size(x))，指定的函数就变为 std2(x)*ones(size(x))。blkproc 函数并不需要结果图像与原图像大小相同，但是如果希望结果图像与原图像大小相同，就必须定义合适的变换函数。

运算结果如图 3-14 所示。



（a）原图像　　（b）结果图像 I1

（c）结果图像 I2　　（d）结果图像 I3

图 3-14　基于不同的图像块划分方式计算图像的局部标准差

3.3.2 滑动邻域操作

在 MATLAB 中，滑动邻域是一个像素集，像素集包含的元素由中心像素的位置决定。滑动邻域操作一次只处理一个图像像素。当操作从图像矩阵的一个位置移动到另一个位置时，滑动邻域也以相同的方向运动，其示意图如图 3-15 所示。

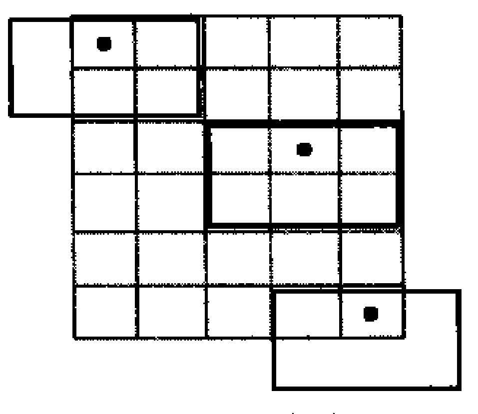
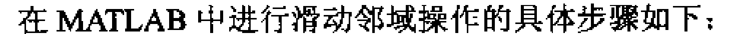


图 3-15 滑动邻域示意图

图 3-15 的滑动邻域是一个 2×3 的矩阵，黑点表示中心像素。对于 m×n 的滑动邻域来说，中心像素的位置是：

floor([(m+1)/2, (n+1)/2])

其中，floor 表示对其参数的每一个分量向下就近取整，因此，对于 2×2 邻域，其中心就是(1,1)，图 3-15 中 2×3 滑动邻域的中心就是(1, 2)。



在 MATLAB 中进行滑动邻域操作的具体步骤如下：

（1）选择像素；

（2）确定该像素的滑动邻域；

（3）调用合适的函数对滑动邻域中的元素进行计算；

（4）将计算结果作为输出图像中对应像素的值；

（5）重复计算，遍及图像中的所有像素。

MATLAB 工具箱提供了一个通用的滑动邻域操作函数 nlfilter。利用 nlfilter 函数可以实现滑动邻域的移动操作，并把待处理图像、邻域大小和一个处理函数（返回值为标量）作为参数，返回与输入图像大小相同的图像作为输出结果。其语法格式为：

```
B = nlfilter(A,[m n],fun)
B = nlfilter(A,[m n],fun,P1,P2,…)
B = nlfilter(A,'indexed',…)
```

B = nlfilter(A,[m n],fun)表示对图像 A 进行操作得到图像 B，其中，[m n]表示滑动邻域的大小为 m×n，fun 是作用于图像邻域上的处理函数。函数 fun 的输入是大小为 m×n 的矩阵，返回值是一个标量值。假定 x 表示某一个图像邻域矩阵，c 表示函数 fun 的返回值，则有表达式 c = fun(x)，c 就表示对应图像邻域 x 的中心像素的输出值。

B = nlfilter(A,[m n],fun,P1,P2,…)可以传递参数 P1,P2,…给函数 fun。

B = nlfilter(A,'indexed',…)把图像 A 作为索引色图像进行处理，如果图像数据是 double 类型，则对其图像邻域进行填补（Padding）时，对图像以外的区域补“1”，而当图像数据为 uint8 类型时，用“0”填补空白区域。

函数 nlfilter 的参数 fun 可以是一个函数句柄或是一个内联函数，看下面的例子。

（1）fun 作为一个函数句柄

```
B = nlfilter(A,[3 3],@myfun);
```

这里，myfun 表示一个 M 文件，其中包含以下语句：

```
function scalar = myfun(x)
scalar = median(x(:));
```

（2）fun 作为一个内联函数

```
fun=inline('median(x(:))');
```

注意：函数 nlfilter 所能支持的数据类型依赖于其参数 fun，即处理函数，返回类型同样由 fun 决定。另外，当被处理图像很大时，函数 nlfilter 往往需要较长的处理时间，此时可以考虑使用快速处理函数 colfilt，在很多情况下，colfilt 都可以完成 nlfilter 同样的处理功能，但处理速度要快得多。对于函数 colfilt 的使用，在下一小节将会进行介绍。

下面举一个实例，计算输入图像的 3×3 邻域（被处理像素本身和其周边的八个邻域在内的九个像素点）像素值的最大值作为输出像素点的像素值。作为参数的处理函数（最大值函数 max）采用内联函数形式，实现代码如下：

```
I = imread('LENA256.bmp');
f = inline('max(x(:))');
I2 = nlfilter(I,[3 3],f);
imshow(I);
figure, imshow(I2)
```

如图 3-16 所示是在执行上述代码时弹出来的状态条，显示执行的进程，整个计算过程用了大约 1 分钟的时间，处理结果如图 3-17 所示。接下来的内容会向读者介绍一个快速进行图像块处理的函数，colfilt，比较完成相同处理任务两者耗费的时间差异，表明后者的计算复杂度明显降低。



图 3-16　在执行邻域操作时的状态条



（a）原图像



（b）结果图像

图 3-17　基于滑动邻域的图像局部最大值计算

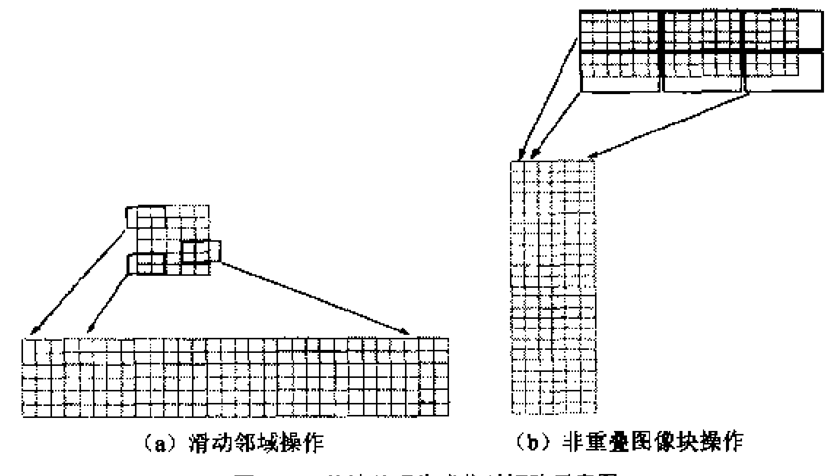
3.3.3　图像块处理的快速算法

前面讲到的非重叠图像块操作和滑动邻域操作都是对原图像进行划分，进而对每个图像块矩阵逐个调用处理函数进行操作。然而，在前面我们举的一个滑动邻域操作的例子，读者可以看到，为了对一幅 256×256 大小的灰度图像执行这一操作，用时大约 1 分钟。可想而知，如果是处理更大的图像，或者是作用更复杂的处理函数，其处理时间会更长。

为了加快图像块处理的速度，MATLAB 图像处理工具箱提供了一个快速处理函数 colfilt，可以处理非重叠图像块操作和滑动邻域操作。该函数在处理过程中，会首先把要处理的图像块

（非重叠块或滑动邻域）重新按列组合成一个临时矩阵，继而对该临时矩阵的每一列（对应着每一个要处理的图像块）调用处理函数。在这一过程中，因为实现了矩阵式操作，大大提高了处理速度，根本没有弹出如图 3-16 所示的状态条，几乎是立刻得到完全一样的计算结果。

如图 3-18 所示，（a）表示利用函数 colfilt 对一个 6×5 大小的图像进行滑动邻域操作，滑动邻域的大小为 2×3，可以看到，所有要处理的滑动邻域被映射到一个临时矩阵，该矩阵的一列就代表一个滑动邻域，从而得到临时矩阵的大小为 6×30；（b）表示利用函数 colfilt 对一个 6×16 大小的图像进行非重叠图像块操作，非重叠图像块的大小为 4×6，将所要处理的图像块映射到一个临时矩阵，矩阵的一列就代表一个非重叠图像块，从而得到临时矩阵的大小为 24×6。



（a）滑动邻域操作　　（b）非重叠图像块操作

图 3-18　快速处理生成临时矩阵示意图

colfilt 函数的语法格式如下：

B = colfilt(A,[m n],block_type,fun)

B = colfilt(A,[m n],block_type,fun,P1,P2,⋯)

B = colfilt(A,[m n],[mblock nblock],block_type,fun,⋯)

B = colfilt(A,'indexed',⋯)

B = colfilt(A,[m n],block_type,fun)表示对图像 A 实现快速的以函数 fun 表示的图像块操作得到输出图像 B，图像块的尺寸为 m×n，block_type 指定了图像块的类型，即

block_type='distinct'，非重叠图像块

block_type='sliding'，滑动邻域

fun 为运算函数，其形式为 y=fun(x)。

B = colfilt(A,[m n],block_type,fun,P1,P2,⋯)指定 fun 中除 x 以外的其他参数 P1、P2、⋯。

B = colfilt(A,[m n],[mblock nblock],block_type,fun,⋯)是一种节省内存的处理方法。从 colfilt 生产临时矩阵的方法，读者可以看到该种处理方法将比寻常的图像块操作方式占用更多的内存。为了既提高图像块处理的速度，又能节省占用的内存，该语句实现把图像 A 分割成 mblock×nblock 大小的子图像，同时只是对该子图像进行操作，通过多次调用函数 fun 来完成整幅图像的操作。这种方法的计算结果与不进行子图像分割是完全一样的。

colfilt 函数生成的临时矩阵被传递给自定义函数，自定义函数为矩阵的每一列返回一个单独的值(标量值)。MATLAB 中很多函数有这种功能，比如 mean、std 和 max 等。返回值赋给输出图像中对应的像素。

下面我们同样采用在滑动邻域操作中介绍的例子来说明函数 colfilt 的应用，并比较处理

速度。实现代码如下：

```
I = imread('LENA256.bmp');
f = inline('max(x)');
I2 = colfilt (I,[3 3], sliding, f);
imshow(I);
figure, imshow(I2)
```



图 3-19　快速处理图像局部最大值计算

处理结果如图 3-19 所示，处理效果与 nlfilter 函数完全一样，因为本来它们的操作函数就是一样的，但处理时间只有大约 1s，相比 nlfilter 函数，colfilt 的处理速度要快得多。

注意：这里 f 的定义是 max(x)，而不是 max(x(:))。这是因为在原图像中，每个像素的邻域已经被排列成列向量。

同样，也可以利用 colfilt 函数对输入图像进行'distinct'类型的块操作，即非重叠图像块操作，处理速度也会比 blkproc 函数要快。然而，colfilt 并不能完全代替 blkproc 函数，而是有以下限制：

- 输出图像必须和输入图像的尺寸相同；
- 图像块不能有重叠，即不能处理包含有“重叠（Overlap）”定义区域的“非重叠（Distinct）”图像块划分方式。

在这两种限制情况之外，就只能调用 blkproc 函数。

接下来介绍另外两个函数，即 im2col 和 col2im。这两个函数通常配套使用，是函数 colfilt 实现图像块到列向量映射的关键。在其他实现快速运算的场合，也可以应用到。

（1）im2col

im2col 函数实现将图像块排列成向量的功能，其语法格式为：

```
B = im2col(A,[m n],block_type)
B = im2col(A,[m n])
B = im2col(A,'indexed',…)
```

这里 B=im2col(A,[m n],block_type)将图像 A 的每一个 m×n 块转换成一列，重新组合成图像 B。

block_type 指定了图像块的排列方式，即

block_type='distinct'，非重叠图像块

block_type='sliding'，滑动邻域

（2）col2im

col2im 函数用于将向量重新排列成图像块。MATLAB 的向量计算功能特别强大，因而通常利用函数 im2col 将图像块转化为列向量，速度会提高很多，处理完之后则需要调用 col2im 函数将列向量重新排列成矩阵。

col2im 语法格式为：

```
A = col2im(B,[m n],[mm nn], block_type)
A = col2im(B,[m n],[mm nn])
```

A = col2im(B,[m n],[mm nn], block_type)表示将图像 B 的每一列重新排列成 m×n 的图像块，用这些图像块组合成 mm×nn 的图像 A，block_type 指定了排列的方式，即

block_type='distinct'，非重叠图像块

block_type='sliding'，滑动邻域

3.4 特定区域处理

在进行图像处理时，有时只需要对图像中的某个特定区域进行处理，而并不需要对整个图像进行处理。比如要对用户选定的一个区域作均值滤波或对比度增强，MATLAB 就可以只对指定的区域进行处理。

3.4.1 指定感兴趣区域

MATLAB 中对特定区域的处理是通过二值掩模来实现的。用户选定一个区域后会生成一个与原图大小相同的二值图像，选定的区域为白色，其余部分为黑色。通过掩模图像就可实现对特定区域的选择性处理。

MATLAB 图像处理工具箱提供了 3 个函数用于生成二值掩模，从而选择特定区域，下面分别介绍。

（1）roipoly

roipoly 函数用于选择图像中的多边形区域。roipoly 函数返回二值图像 BW，选中区域的像素值为 1，其余部分的值为 0。这个二值图像可以作为掩模，通过与原图的运算选择目标或背景。其语法格式如下：

```
BW = roipoly(I,c,r)
BW = roipoly(I)
BW = roipoly(x,y,I,xi,yi)
[BW,xi,yi] = roipoly(…)
[x,y,BW,xi,yi] = roipoly(…)
```

BW=roipoly(I,c,r)是用向量 c、r 指定多边形各角点的 x、y 轴的坐标。

BW=roipoly(I)是允许用户交互选择多边形区域，选择角点，用空格键和 Del 键撤销选择，按 Enter 键确认选择。

BW=roipoly(x,y,I,xi,yi)用矢量 x 和 y 建立非默认的坐标系，然后在指定的坐标系下选择由向量 xi、yi 指定的多边形区域。

[BW,xi,yi]=roipoly(…)交互选择多边形区域，并返回多边形角点的坐标。

[x,y,BW,xi,yi]=roipoly(…)交互选择多边形区域后，还返回多边形顶点在指定为坐标系 x–y 下的坐标。

下面的例子将根据指定的坐标选择一个六边形区域，结果如图 3-20 所示。

```
I = imread('eight.tif');
c = [222 272 300 270 221 194];
r = [21 21 75 121 121 75];
BW = roipoly(I,c,r);
imshow(I)
```

figure, imshow(BW)

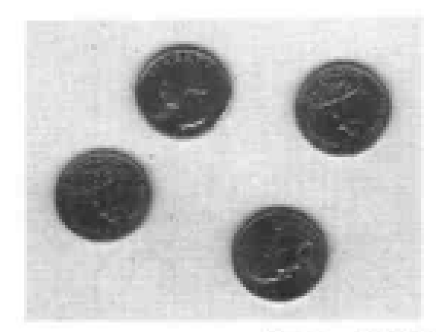
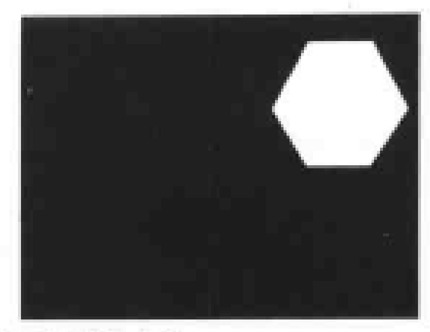



图 3-20　根据指定的坐标选择六边形

（2）roicolor

MATLAB 图像处理工具箱提供的 roicolor 函数可以对 RGB 图像和灰度图像实现按灰度或亮度值选择区域，其语法格式为：

BW = roicolor(A,low,high)

BW = roicolor(A,v)

BW = roicolor(A,low,high)表示按指定的灰度范围分割图像，返回二值掩模 BW，[low high]为所要选择区域的灰度范围。如果 low 大于 high，则返回为空矩阵。

BW=roicolor(A,v)是按向量 v 中指定的灰度值来选择区域。

下面的例子是按灰度分割图像中的目标，结果如图 3-21 所示。

```
I = imread('rice.png');
BW = roicolor(I,128,255);
imshow(I);
figure, imshow(BW)
```

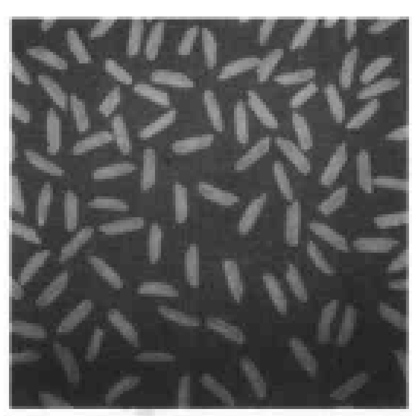



图 3-21　按照指定灰度范围选择图像区域结果

（3）poly2mask

该函数可以转化指定的多边形区域为二值掩模。语句格式为

BW = poly2mask(x,y,m,n)

x 和 y 表示两个向量，指定一个多边形区域，BW 的大小为 m×n，在 BW 中指定区域内的像素为 1，指定区域外的像素为 0。如果 x 和 y 指定的区域不封闭，则 poly2mask 自动封闭这个多边形。下面的例子给出了函数 poly2mask 的调用格式。

x = [63 186 54 190 63];

```
y = [60 60 209 204 60];
bw = poly2mask(x,y,256,256);
imshow(bw)
hold on
plot(x,y,'b','LineWidth',2)
hold off
```

3.4.2 特定区域滤波

MATLAB 图像处理工具箱中提供了一个区域滤波函数 roifilt2，其语法格式为：

```
J = roifilt2(h,I,BW)
J = roifilt2(I,BW,fun)
J = roifilt2(I,BW,fun,P1,P2,…)
```

J = roifilt2(h,I,BW)使用滤波器 h 对图像 I 中用二值掩模 BW 选中的区域进行滤波。

J=roifilt2(I,BW,fun)和 J=roifilt2(I,BW,fun,P1,P2,…)对图像 I 中用二值掩模 BW 选中的区域作函数运算 fun，其中 fun 是描述函数运算的字符串，参数为 p1、P2、……返回图像 J 在选中区域的像素为图像 I 经 fun 运算的结果，其余部分的像素值为 I 的原始值。

下面的例子是对指定区域进行锐化滤波，结果如图 3-22 所示。

```
I = imread('eight.tif');
c = [222 272 300 270 221 194];
r = [21 21 75 121 121 75];
BW = roipoly(I,c,r);%指定滤波区域为 c 和 r 确定的多边形
h = fspecial('unsharp');%指定滤波算子为 unsharp
J = roifilt2(h,I,BW);
imshow(J), figure, imshow(J)
```

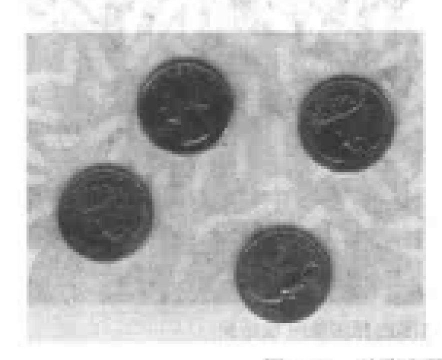
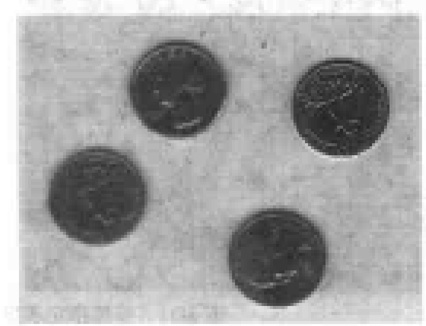



图 3-22　对选定区域进行滤波的结果

从图像中可以看出，右上角的硬币发生变化，而其他硬币保持不变。由此可知，roifilt2 函数只对指定的区域进行滤波。

3.4.3 特定区域填充

MATLAB 图像处理工具箱中提供了函数 roifill 用于对特定区域的填充，其语法格式为：

```
J = roifill(I,c,r)
J = roifill(I)
J = roifill(I,BW)
[J,BW] = roifill(…)
J = roifill(x,y,I,xi,yi)
[x,y,J,BW,xi,yi] = roifill(…)
```

其中 J=roifill(I,c,r)填充由向量 c、r 指定的多边形，c 和 r 分别为多边形各顶点的 *x*、*y* 坐标。它是通过求解边界的拉普拉斯方程，利用多边形边界点的灰度平滑的插值得到多边形内部的点。通常可以利用对指定区域的填充来“擦”掉图像中的小块区域。

J=roifill(I)表示由用户交互选取填充的区域。选择多边形的角点后，按 Enter 键表示结束，空格键或 Del 键表示取消一个选择。

J=roifill(I,BW)用掩模图像 BW 选择区域。

[J,BW]=roifill(…)在填充区域的同时还返回掩模图像 BW。

J=roifill(x,y,I,xi,yi)和[x,y,J,BW,xi,yi]=roifill(…)表示在指定的坐标系 *x*–*y* 下填充由向量 xi 和 yi 指定的多边形区域。

下面的例子为填充指定的区域，结果如图 3-23 所示。

```
I = imread('eight.tif');
c = [222 272 300 270 221 194];
r = [21 21 75 121 121 75];
J = roifill(I,c,r);
imshow(I)
figure, imshow(J)
```

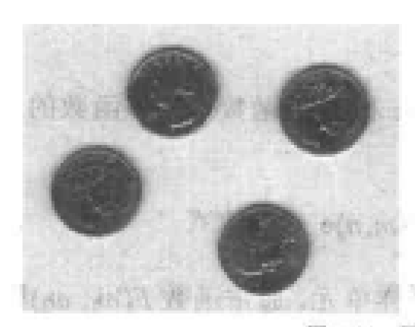
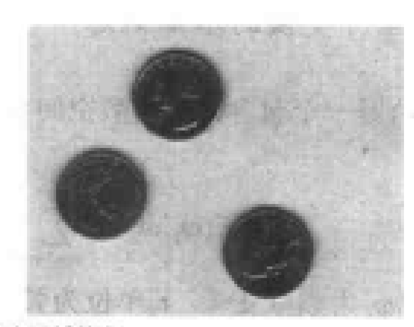



图 3-23　图像指定区域填充

第4章 图 像 变 换

在 MATLAB 中，一般用二元函数 $f(x,y)$ 作为图像的数学表示。$f(x,y)$ 表示在特定点 (x,y) 处的函数值，表示图像在该点相应的颜色强度或者灰度。所谓图像变换就是指把图像转换为另一种数学表示方式的操作。

在图像处理技术中，图像的正交变换技术有着广泛的应用，是图像处理的重要工具。通过变换图像，改变图像的表示域及表示数据，可以给后继工作带来极大的方便。例如，傅立叶变换可使处理分析在频域中进行，使运算简单；而离散余弦变换（DCT）可使能量集中在少数数据上，从而实现数据压缩，便于图像传输和存储。

在 MATLAB 图像处理工具箱中，提供了几种常用的图像变换函数，它们是傅立叶变换（Fourier Transform）、离散余弦变换（Discrete Cosine Transform）和 Radon 变换（Radon Transform）。另外，随着小波分析方法在图像处理中的应用不断发展成熟，MATLAB 小波分析工具箱也提供了很多小波变换的函数，用于图像处理。鉴于篇幅限制，关于小波在图像处理中的应用请读者参考 MATLAB 辅助小波变换方面的书籍。

4.1 傅立叶变换

4.1.1 傅立叶变换的基本概念

假设 $f(m,n)$ 是一个包含两个离散空间变量 m 和 n 的函数，则该函数的二维傅立叶变换的定义如下：

$$F(\omega_1,\omega_2)=\sum_{m=-\infty}^{\infty}\sum_{n=-\infty}^{\infty}f(m,n)\mathrm{e}^{-\mathrm{j}\omega_1 m}\mathrm{e}^{-\mathrm{j}\omega_2 m} \tag{4.1}$$

式中，ω_1 和 ω_2 为频域变量，其单位为弧度/采样单元。通常函数 $F(\omega_1, \omega_2)$ 称为函数 $f(m, n)$ 的频域表示。$F(\omega_1, \omega_2)$ 是复变函数，其变量 ω_1 和 ω_2 的周期均为 2π。因为这种周期性的存在，所以通常在图像显示时，这两个变量的取值范围为 $-\pi \leqslant \omega_1,\omega_2 \leqslant \pi$。

傅立叶反变换定义如下：

$$f(m,n)=\frac{1}{4\pi^2}\int_{-\pi}^{\pi}\int_{-\pi}^{\pi}F(\omega_1,\omega_2)\mathrm{e}^{\mathrm{j}\omega_1 m}\mathrm{e}^{\mathrm{j}\omega_2 m}\mathrm{d}\omega_1\mathrm{d}\omega_2 \tag{4.2}$$

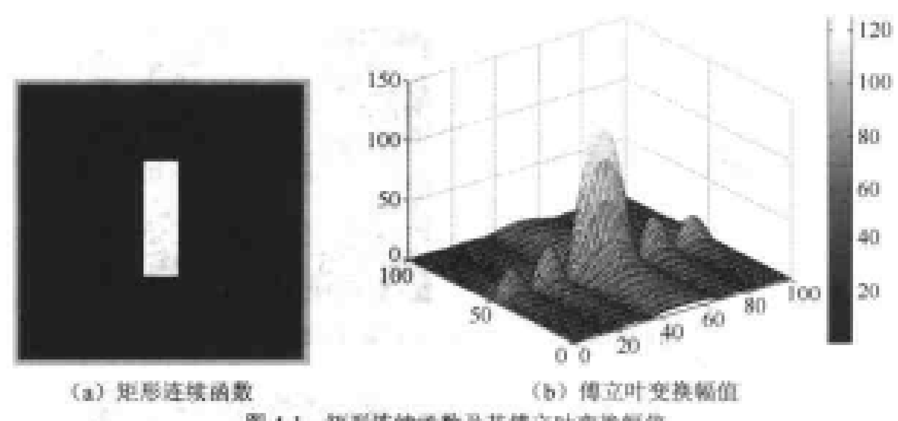
简单地说，该方程说明 $f(m, n)$ 可以表示为无限多项不同频率的复指函数之和。而不同的频率点 (ω_1, ω_2) 所做的贡献由幅度 $F(\omega_1, \omega_2)$ 決定。

例如，考察下面的矩形函数 $f(m, n)$。该函数在一个矩形的区域中的函数值为 1，其他区域都为 0，如图 4-1（a）所示。

在 MATLAB 中，变量 m、n 和函数 $f(m, n)$ 均采用离散表示，所以要想真实地逼近连续函数，只能通过提高取样率。因此，$f(m, n)$ 函数的傅立叶变换可由以下程序段获得，傅立叶变

换的幅值即$|F(\omega_1, \omega_2)|$，如图 4-1（b）所示。其中 x 轴和 y 轴分别为水平分量和垂直分量。

```
clear
N=100
f=zeros(50,50);
f(15:35,23:28)=1;
figure(1)
imshow(f,'notruesize')
F=fft2(f,N,N);
F2=fftshift(abs(F));
figure(2)
x=1:N;y=1:N;
mesh(x,y,F2(x,y));
colormap(gray);colorbar
```



（a）矩形连续函数　　（b）傅立叶变换幅值

图 4-1　矩形连续函数及其傅立叶变换幅值

对于傅立叶变换结果，通常还采用另一种方法进行显示，即将变换结果的函数值取对数，即 $\log|F(\omega_1, \omega_2)|$，这种显示方式可以使得$|F(\omega_1, \omega_2)|$接近于 0 值部分的细节凸现出来，如图 4-2 所示。

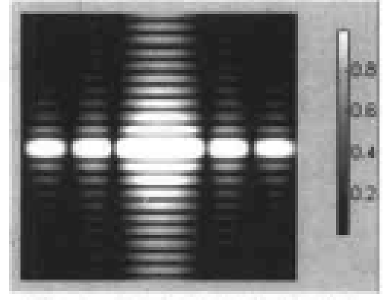
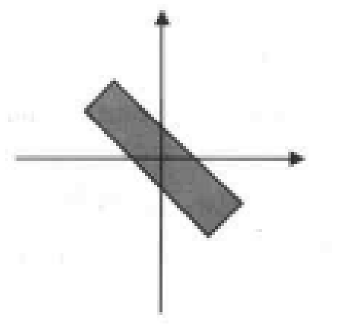
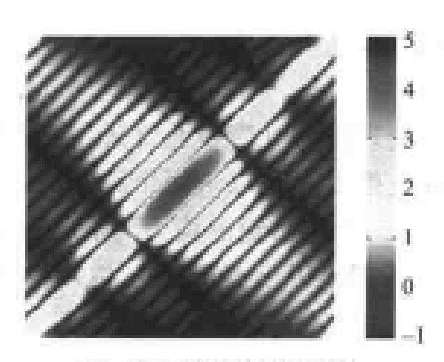


图 4-2　傅立叶变换幅值对数图形

对如图 4-1 所示的矩形函数旋转一个角度，可得到图 4-3a，然后对其进行傅立叶变换，得到如图 4-3b 所示的幅值对数图。

（a）旋转后的矩形函数　　（b）傅立叶变换幅值对数图

图 4-3　矩形函数及其傅立叶变换

比较图 4-2 和图 4-3，可以发现二维傅立叶变换具有如下旋转性质：

如果在极坐标下表示二维函数图形，把空间域和空间频域的直角坐标均作坐标转换，

空间域　$x = r\cos\theta, y = r\sin\theta$

空间频域　$u = \omega\cos\phi, y = \omega\sin\phi$

则有 $f(m, n)$ 在空间域极坐标系中表示为 $f(r, \theta)$，$F(u, v)$ 在空间频域极坐标系中表示为 $F(\omega, \phi)$。如果有

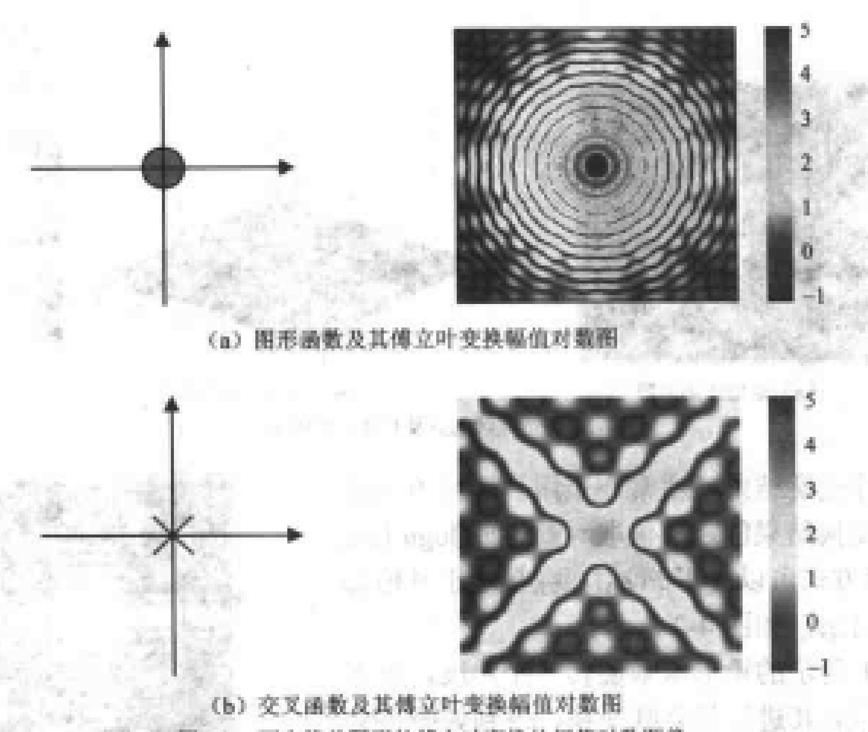
$$f(r,\theta) \Leftrightarrow F(\omega,\phi)$$

则有

$$f(r,\theta+\theta_0) \Leftrightarrow F(\omega,\phi+\phi_0)$$

即如果 $f(x, y)$ 在空间域旋转一个角度 θ_0 后得到新的函数 $f(r,\theta+\theta_0)$，其对应的傅立叶变换 $F(\omega,\phi+\phi_0)$ 是 $f(x, y)$ 的傅立叶变换 $F(\omega, \phi)$ 在空间频域中旋转同样的角度 θ_0 得到的函数。反之亦然。

下面列举另外两个简单图形的傅立叶变换的幅值对数的图像，如图 4-4 所示。



（a）图形函数及其傅立叶变换幅值对数图

（b）交叉函数及其傅立叶变换幅值对数图

图 4-4　两个简单图形的傅立叶变换的幅值对数图像

4.1.2　离散傅立叶变换

在用计算机处理傅立叶变换通常采用离散傅立叶变换（Discrete Fourier Transform，DFT）。采用离散傅立叶变换主要有以下两个原因：

（1）因为 DFT 的输入/输出均为离散值，非常适用于计算机的运算操作。

（2）采用离散傅立叶变换，就可以用一种快速算法，即快速傅立叶变换（Fast Fourier Transform，FFT）。

FFT 的设计思想是将原函数分为奇数项和偶数项，通过不断将一个奇数项和一个偶数项相加（减），得到需要的结果。

也就是说 FFT 是将复杂的乘法运算变成两个数相加（减）的简单运算的重复，即通过计算两个单点的 DFT，来计算两个双点的 DFT：通过计算两个双点的 DFT，来计算四个点的 DFT……依此类推。

对于任何 $N=2^m$ 的 DFT 的计算，通过计算两个 $N/2$ 点的 DFT，来计算 N 个点的 DFT。

设离散函数 $f(m,n)$在有限区域 $0\leqslant m\leqslant M-1$，$0\leqslant n\leqslant N-1$ 非零。则 2-D 的 $M\times N$ FFT 和 $M\times N$ 逆 FFT 关系推导过程如下：

令

$$W_N^{pm}=\exp(-\mathrm{j}\frac{2\pi pm}{N}) \tag{4.3}$$

则有

$$\begin{aligned}F(p)&=\frac{1}{N}\sum_{m=0}^{N-1}f(m)W_N^{pm}\\&=\frac{1}{2}\left[\frac{2}{N}\sum_{m=0}^{N/2-1}f(2m)W_N^{2pm}+\frac{2}{N}\sum_{m=1}^{N/2-1}f(2m+1)W_N^{p(2m+1)}\right]\\&=\frac{1}{2}\left[\frac{1}{M}\sum_{m=0}^{M-1}f(2m)W_N^{pm}+\frac{1}{M}\sum_{m=1}^{M-1}f(2m+1)W_N^{pm}W_N^{p}\right]\\&=\frac{1}{2}\left[F_{\mathrm{e}}(p)+W_N^{p}F_{\mathrm{o}}(p)\right]\end{aligned} \tag{4.4}$$

上式中在计算时分成了奇数项和偶数项。

同理

$$\begin{aligned}F(p+M)&=\frac{1}{2}\left[F_{\mathrm{e}}(p+M)+W_N^{p+M}F_{\mathrm{o}}(p+M)\right]\\&=\frac{1}{2}\left[F_{\mathrm{e}}(p)+W_N^{p+M}F_{\mathrm{o}}(p)\right]\end{aligned} \tag{4.5}$$

又

$$\begin{aligned}W_N^{p+M}&=W_N^{p}\cdot W_N^{M}=W_N^{p}\cdot\exp(-\mathrm{j}\frac{2\pi M}{N})\\&=W_N^{p}\cdot\exp(-\mathrm{j}\pi)=-W_N^{p}\end{aligned} \tag{4.6}$$

所以

$$F(p+M)=\frac{1}{2}\left[F_{\mathrm{e}}(p)-W_N^{p}F_{\mathrm{o}}(p)\right]$$

由以上两式推导可得 FFT 的定义式为

$$F(p,q)=\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}f(m,n)\mathrm{e}^{-\mathrm{j}(2\pi/M)pm}\mathrm{e}^{-\mathrm{j}(2\pi/N)qn}\qquad\begin{aligned}p&=0,1,\ldots,M-1\\q&=0,1,\ldots,N-1\end{aligned} \tag{4.7}$$

$$f(m,n)=\frac{1}{MN}\sum_{p=0}^{M-1}\sum_{q=0}^{N-1}F(p,q)\mathrm{e}^{\mathrm{j}(2\pi/M)pm}\mathrm{e}^{\mathrm{j}(2\pi/N)qn}\qquad\begin{aligned}m&=0,1,\ldots,M-1\\n&=0,1,\ldots,N-1\end{aligned} \tag{4.8}$$

可以认为，$F(p,q)$是 $f(m,n)$的 DFT 系数。

在 MATLAB 中，可分别用函数 fft、fft2 和 fftn 来计算一维、二维和 n 维的 FFT，而其反变换依次为 ifft、ifft2 和 ifftn。

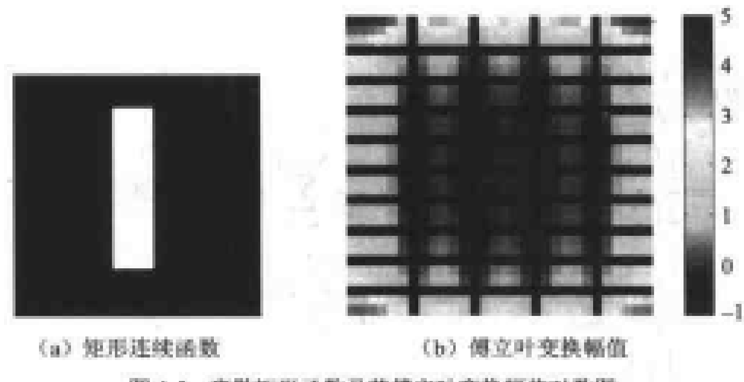
下面构建一个类似于图 4-1 所示的矩形函数，代码如下：

```
f = zeros(30,30);
f(5:24,13:17) = 1;
imshow(f,'notruesize')
```

函数结果显示如图 4-5（a）所示。然后对 f 进行二维快速傅立叶变换，代码如下：

```
F = fft2(f);
F2 = log(abs(F));
imshow(F2,[-1 5],'notruesize'); colormap(jet); colorbar
```

变换结果如图 4-5（b）所示。



（a）矩形连续函数　　（b）傅立叶变换幅值

图 4-5　离散矩形函数及其傅立叶变换幅值对数图

比较图 4-5（b）和图 4-2 可以发现图 4-2 具有更高的分辨率，并且图 4-2 中的零频率分量显示在图像中央，而图 4-5（b）中显示在左上角。

针对离散傅立叶变换分辨率低和零频率分量显示区域调整的问题，MATLAB 分别提供了补零操作和改变图像显示象限的函数 fftshift。

在计算离散傅立叶变换时，可以通过下面的语句对被变换函数（或图像区域）进行补零操作来提高分辨率，具体调用方式如下：

```
F = fft2(f,M,N);
```

$M \times N$ 表示进行补零操作后被变换区域的大小，与原区域不相重叠的区域补零。例如，我们设定补零后的区域大小为 256×256，进行下面的计算：

```
F = fft2(f,256,256);
imshow(F2,[-1 5],'notruesize');
colormap(jet); colorbar
```

结果如图 4-6 所示。事实上，了解快速傅立叶变换的读者应该知道，在执行 FFT 运算时，要求参与计算的点数必须是 2 的整数次方，不足的则补零。所不同的是，FFT 执行的是向上就近取 2 的整数次方，而 F = fft2(f,256,256)是由用户设定变换区域的大小。当然，如果用户设定的区域大小不是 2 的整数次方，FFT 还会有默认的补零操作。

为使变换结果的零频率分量位于中心，函数 fftshift 采用如下的调用方式：

```
F = fft2(f, 256, 2566)
F2 = fftshift(F);
imshow(log(abs(F2)),[.1 5]);
colormap(jet);colorbar;
```

执行结果如图 4-7 所示。

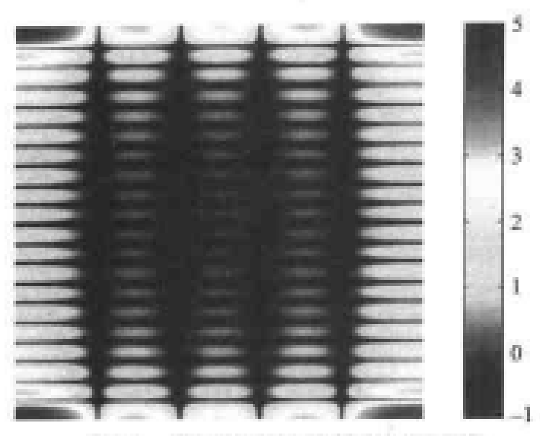


图 4-6 具有补零操作的离散傅立叶变换

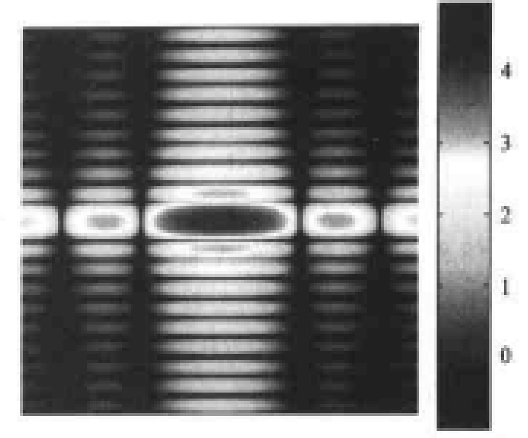


图 4-7 移位操作后的变换结果（零频率分量在中心）

4.1.3 傅立叶变换函数的应用

本节介绍傅立叶变换在图像处理中应用的几个例子。

1．线性滤波器频率响应

由线性系统理论可知，滤波器冲激响应的傅立叶变换就是该滤波器的频率响应。MATLAB工具箱中提供的 freqz2 函数可以同时计算和显示滤波器的频率响应。高斯卷积核的频率响应如图 4-8 所示，它的频率响应体现了高斯函数的低通特性和高频衰减特性。

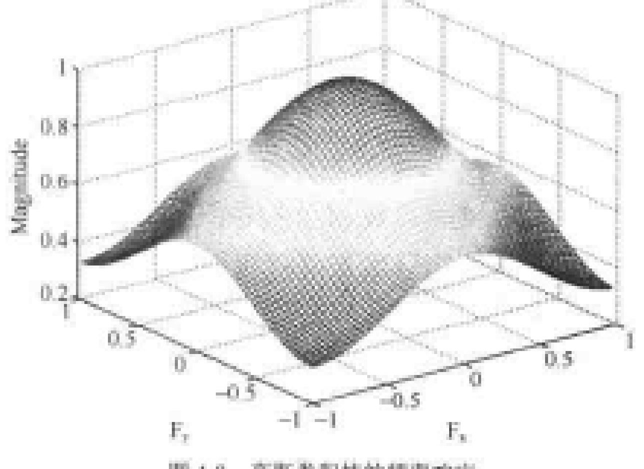


图 4-8 高斯卷积核的频率响应

2．快速卷积

傅立叶变换的另一个重要特性是能够实现快速卷积。由线性系统理论可知，两个函数的卷积的傅立叶变换等于两个函数的傅立叶变换的乘积。该特性与快速傅立叶变换一起，可以快速计算函数的卷积。

假设 A 为 M×N 的矩阵，B 为 P×Q 的矩阵，则快速计算卷积的方法如下：

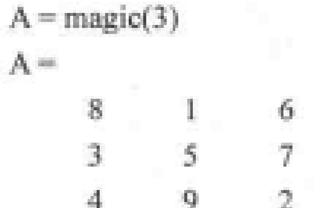


对 A 和 B 补零，使其大小都为(M+P−1)×(N+Q−1)。

利用 fft2 函数对 A 和 B 分别进行二维 FFT 变换。

将两个 FFT 结果相乘，利用函数 ifft2 对乘积进行傅立叶反变换。

举例如下：



```
A = magic(3)
A =
     8     1     6
     3     5     7
     4     9     2
```

```
B = ones(3)
B =
      1     1     1
      1     1     1
      1     1     1
A(8,8) = 0;%对矩阵 A 补零
B(8,8) = 0;%对矩阵 A 补零
C = ifft2(fft2(A).*fft2(B));%正变换与反变换结合
C = C(1:5,1:5);
C = real(C)
```

程序执行的结果为

```
C =
    8.0000    9.0000   15.0000    7.0000    6.0000
   11.0000   17.0000   30.0000   19.0000   13.0000
   15.0000   30.0000   45.0000   30.0000   15.0000
    7.0000   21.0000   30.0000   23.0000    9.0000
    4.0000   13.0000   15.0000   11.0000    2.0000
```

MATLAB 工具箱中提供的二维卷积函数为 conv2，读者可以对比两种实现方式执行的结果和速度。代码如下：

```
c=con2(A,B)
c=c(1:5,1:5)
```

3. 图像特征识别

傅立叶变换可以用于与卷积密切相关的相关运算（correlation）。在数字图像处理中的相关运算通常用于匹配模板，可以用于对某些模板对应的特征进行定位。例如，假如我们希望在图像 text.tif 中定位字母“a”，如图 4-9 所示，可以采用下面的方法定位。

将包含字母“a”的图像与图像 text.png 进行相关运算，也就是对字母“a”的图像和图像 text.png 进行傅立叶变换，然后利用快速卷积的方法，计算字母“a”和图像 text.png 的卷积，提取卷积运算的峰值，即得到在图像 text.png 中对应字母“a”的定位结果。执行程序如下：

```
bw = imread('text.png');
a = bw(32:45,88:98);
imshow(bw);
figure, imshow(a);
C = real(ifft2(fft2(bw) .* fft2(rot90(a,2),256,256)));
figure, imshow(C,[])
max(C(:))
ans =
      68.0000
thresh = 60;
```

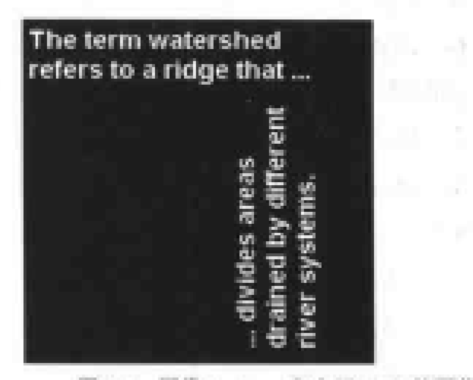


图 4-9 图像 text.png 和字母“a”的图像

```
figure, imshow(C > thresh)
```

程序执行结果如图 4-10 和图 4-11 所示。



图 4-10　快速卷积结果

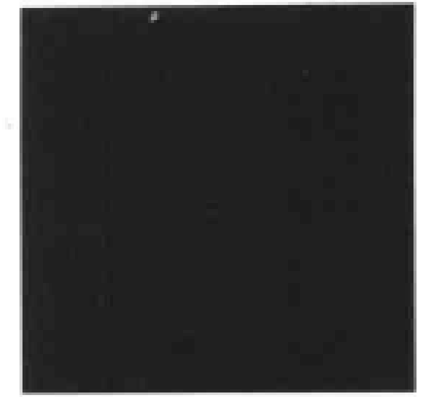


图 4-11　对字母"a"的定位结果（白色小亮点）

4.2　离散余弦变换

在 MATLAB 图像处理工具箱中，dct2 函数用于计算图像的二维离散余弦变换（Discrete Cosine Transform）简称 DCT。大多数情况下，DCT 用于压缩图像，JPEG 图像格式就采用了 DCT 算法。

4.2.1　离散余弦变换的基本概念

假设矩阵 A 的大小为 M×N，其二维离散余弦变换的定义为：

$$B_{p,q}=a_p a_q\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}A_{mn}\cos\frac{\pi(2m+1)p}{2M}\cos\frac{\pi(2n+1)p}{2N}\quad \begin{matrix}0\leqslant p\leqslant M-1\\ 0\leqslant q\leqslant N-1\end{matrix}\tag{4.9}$$

$$a_p=\begin{cases}1/\sqrt{M}\ ,\ p=0\\ \sqrt{2/M}\ ,\ 1\leqslant q\leqslant M-1\end{cases}\qquad a_q=\begin{cases}1/\sqrt{N}\ ,\ q=0\\ \sqrt{2/N}\ ,\ 1\leqslant q\leqslant N-1\end{cases}$$

其中 B_{pq} 称为矩阵 A 的 DCT 系数。在 MATLAB 中，矩阵的下标从 1 开始而不是从 0 开始，所以 A(1,1)和 B(1,1)分别代表上面的 A_{00} 和 B_{00}。

DCT 是一种可逆变换，它的逆变换定义为：

$$A_{mn}=\sum_{p=0}^{M-1}\sum_{q=0}^{N-1}a_p a_q B_{pq}\cos\frac{\pi(2m+1)p}{2M}\cos\frac{\pi(2n+1)p}{2N}\quad \begin{matrix}m=0,1,\ldots,M-1\\ n=0,1,\ldots,N-1\end{matrix}\tag{4.10}$$

$$a_p=\begin{cases}1/\sqrt{M}\ ,\ p=0\\ \sqrt{2/M}\ ,\ 1\leqslant q\leqslant M-1\end{cases}\qquad a_q=\begin{cases}1/\sqrt{N}\ ,\ q=0\\ \sqrt{2/N}\ ,\ 1\leqslant q\leqslant N-1\end{cases}$$

上式的含义是任何 M×N 的矩阵 A 都可以表示为一系列函数的和：

$$a_p a_q B_{pq}\cos\frac{\pi(2m+1)p}{2M}\cos\frac{\pi(2n+1)p}{2N}\quad \begin{matrix}p=0,1,\ldots,M-1\\ q=0,1,\ldots,N-1\end{matrix}\tag{4.11}$$

这些函数称为 DCT 的基函数。DCT 系数 B_{pq} 则被看作是每个基函数的权。对于 8×8 大小的矩阵，其 DCT 的 64 个基函数如图 4-12 所示。

水平方向频率由左向右增加，垂直方向频率由上到下增加。常数值的基函数位于图像的左上角，被称为直流基函数，对应的 DCT 系数 B_{00} 常被称为是直流系数。

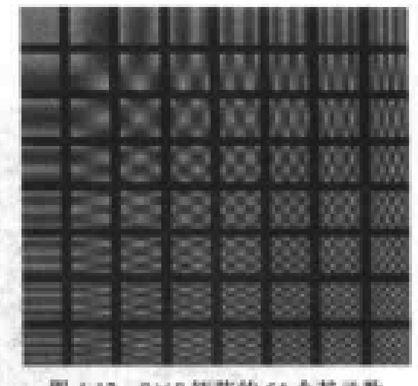


图 4-12　8×8 矩阵的 64 个基函数

MATLAB 图像处理工具箱提供的 DCT 函数有 3 个，分别是 dct2、dctmtx 和 idct2。

（1）dct2

dct2 函数采用基于 FFT 的算法，主要用于实现较大输入矩阵的离散余弦变换。其语法格式为：

B = dct2(A)

B = dct2(A,m,n)

B = dct2(A,[m n])



B=dct2(A)返回图像 A 的二维离散余弦变换值，它的大小与 A 相同，且各元素为离散余弦变换的系数 B(kl,k2)。

B = dct2(A,m,n)或 B = dct2(A,[m n])表示在对图像 A 进行二维离散余弦变换之前，先将图像 A 补零至 m×n。如果 m 和 n 比图像 A 小，则进行变换之前，将图像 A 进行剪切。

（2）dctmtx

dctmtx 函数主要用于实现较小输入矩阵的离散余弦变换。其语法格式为：

D = dctmtx(M)

D = dctmtx(M)返回 M×M 大小的 DCT 矩阵。其形式为

$$D_{pq}=\begin{cases}1/\sqrt{M} & p=0\ ,\ 0\leqslant q\leqslant M-1\\ \sqrt{2/M}\cos\dfrac{\pi(2q+1)p}{2M} & 1\leqslant p\leqslant M-1\ ,\ 0\leqslant q\leqslant M-1\end{cases}\tag{4.12}$$

设 A 是一个 M×M 大小的矩阵，则 D*A 表示 A 的列向量的一维离散余弦变换，而 D'*A（D'表示 D 的转置）表示 A 的列向量的一维逆离散余弦变换。要实现 A 的二维离散余弦变换，只需计算 D*A*D'。这种计算有时会比利用函数 dct2 更快，特别是计算大量小的相同尺寸 DCT 时，矩阵 D 只需计算一次，因而速度快。例如，在实现 JPEG 压缩时，要多次实现大小为 8×8 的图像块的 DCT，为了实现这种变换，首先采用函数 dctmtx 得到矩阵 D，即利用语句 D=dctmtx(8)，然后，对每一个图像块执行运算 B=D*A*D'。由于变换矩阵 D 是实正交矩阵，为此二维逆离散余弦变换为 A= D'*B*D。这种实现方法比调用函数 dct2 要快。

（3）idct2

idct2 函数可以实现图像的二维逆离散余弦变换，其语法格式为：

B = idct2(A)

B = idct2(A,m,n)

B = idct2(A,[m n])

B = idct2(A)计算矩阵 A 的二维逆离散余弦变换，返回图像 B 的大小与 A 相同。

B=idct2(A,m,n)或 H=idct2(A, [m n])表示在对矩阵 A 进行二维逆离散余弦变换之前，先将矩阵 A 补零至 m×n。如果 m 和 n 比矩阵 A 小，则进行变换之前，先对矩阵 A 进行剪切操作，返回图像大小为 m×n。

下面的例子将如图 4-13 所示的一幅图像进行离散余弦变换，可以注意到图像能量的绝大部分位于变换矩阵的左上角，结果如图 4-14 所示。然后，将 DCT 变换值小于 10 的系数设为

0，然后利用 idct2 函数重构图像，其结果如图 4-15 所示。

```
RGB = imread('autumn.tif');
I = rgb2gray(RGB);
J = dct2(I);
imshow(log(abs(J)),[]), colormap(jet(64)), colorbar
J(abs(J) < 10) = 0;
K = idct2(J);
figure,imshow(I)
figure,imshow(K,[0 255])
```



图 4-13　原图像

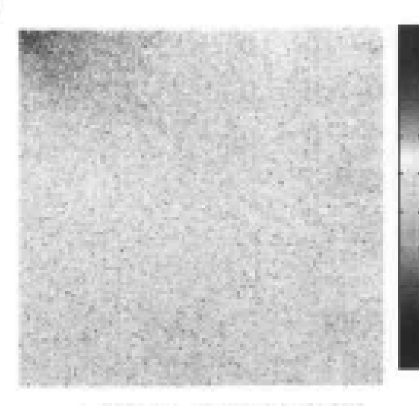


图 4-14　执行 DCT 的结果



图 4-15　对压缩后的图像重构的结果

4.2.2　离散余弦变换函数的应用

在 JPEG 图像压缩算法中，图像常被分成 8×8 或 16×16 的图像块，然后对每个图像块进行离散余弦变换，对变换结果进行量化、编码及传输。在接收端，量化的 DCT 系数被解码，并用来计算每个图像块的逆离散余弦变换，最后把各图像块拼接起来构成一幅图像。对一幅典型的图像而言，许多 DCT 的系数近似为 0，把它们去掉并不会明显影响重构图像的质量。

正如前面介绍的，实现 DCT 的方法有两种，而利用函数 dctmtx 的方法可以比较快速地解决 JPEG 图像压缩问题。下面举一个图像压缩的例子。

在该例中，将输入图像（如图 4-16）划分成 8×8 的图像块，计算它们的 DCT 系数，并只保留 64 个系数中的 10 个，然后利用这 10 个系数对每个图像块实现逆 DCT 来重构图像，结果如图 4-17 所示。



图 4-16　原始图像



图 4-17　压缩后重构图像

```
I = imread('cameraman.tif');
I = im2double(I);
T = dctmtx(8);%产生 DCT 变换矩阵
B = blkproc(I,[8 8],'P1*x*P2',T,T');     %计算二维 DCT
mask = [1   1   1   1   0   0   0   0
        1   1   1   0   0   0   0   0
        1   1   0   0   0   0   0   0
        1   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0];
%二值掩模，用来压缩 DCT 的系数
B2 = blkproc(B,[8 8],'P1.*x',mask);
%只保留 DCT 的 10 个系数
I2 = blkproc(B2,[8 8],'P1*x*P2',T',T);
%逆 DCT，用来重构图像
imshow(I), figure, imshow(I2)
```

4.3 Radon 变换

4.3.1 Radon 变换及其应用

图像处理工具箱的 Radon 函数用来计算指定方向上图像矩阵的投影，二元函数$f(x,y)$的投影是在某一方向上的线积分，例如$f(x,y)$在垂直方向上的线积分是$f(x,y)$在 x 方向上的投影，在水平方向上的积分是在 y 方向上的投影。图 4-18 是一个简单的二维函数在水平垂直方向的投影。

推而广之，投影可沿任意角度θ进行，通常$f(x,y)$的 Radon 变换是$f(x,y)$平行于 y' 轴的线积分，格式如下：

$$R_\theta(x') = \int_{-\infty}^{\infty} f(x'\cos\theta - y'\sin\theta, x'\sin\theta + y'\cos\theta)\mathrm{d}y' \tag{4.13}$$

其中，

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

图 4-19 表示任意角度 Radon 变换的几何关系。

下面的命令计算图像 I 在 theta 矢量指定的方向上的 Radon 变换。

[R,xp] = radon(I,theta)

R 的各行返回 theta 中各方向上的 Radon 变换值，xp 矢量表示沿 x' 轴相应的坐标值。图像 I 的中心在 floor((size(I)+1)/2)，在 x' 轴上对应 $x'=0$。

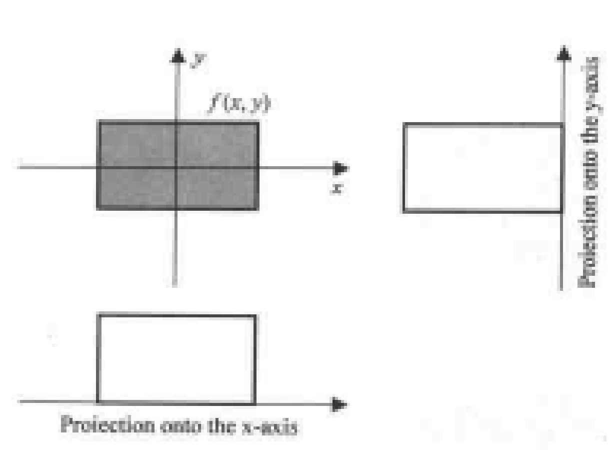


图 4-18 矩形函数在水平垂直方向上的投影

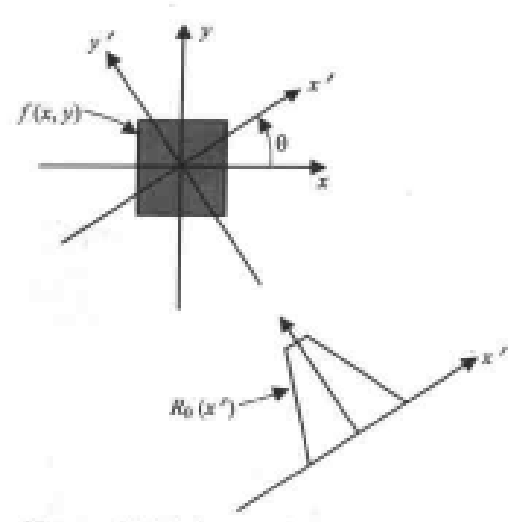
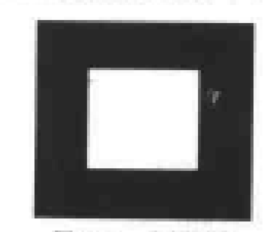


图 4-19 任意角度 Radon 变换的几何关系

计算并画出如图 4-20 所示的含有正方形图像在 0°和 45°方向上的 Radon 变换命令如下：

```
I = zeros(100,100);
I(25:75, 25:75) = 1;
imshow(I)
[R,xp] = radon(I,[0 45]);
figure; plot(xp,R(:,1)); title('R_{0^o} (x\prime)')
figure; plot(xp,R(:,2)); title('R_{45^o} (x\prime)')
```


图 4-20 方框图像

方框图像在 0°和 45°的 Radon 变换如图 4-21 所示。

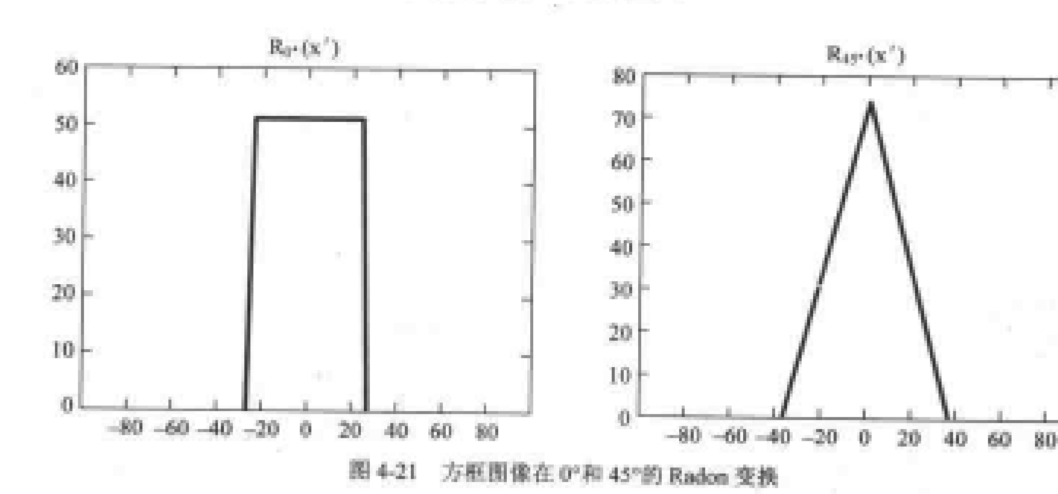


图 4-21 方框图像在 0°和 45°的 Radon 变换

对于有很多角度的 Radon 变换通常可以表示为图像。计算方形图像从 0°到 180°每隔 1°计算一次 Radon 变换的命令如下，其结果如图 4-22 所示。

```
I = zeros(100,100);
I(25:75, 25:75) = 1;
theta = 0:180;
[R,xp] = radon(I,theta);
imagesc(theta,xp,R);
title('R_{\theta} (X\prime)');
xlabel('\theta (degrees)'); ylabel('X\prime');
```

```
set(gca,'XTick',0:20:180);
colormap(hot); colorbar
```

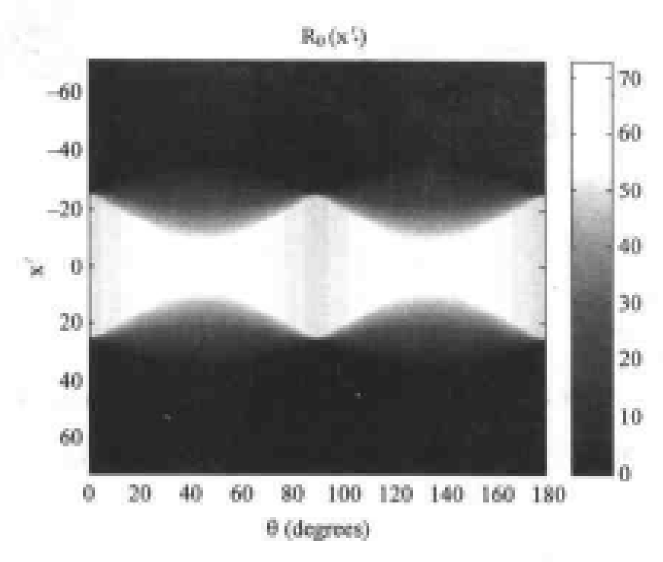


图 4-22　方框图像在 0°~180° 的 Radon 变换

我们知道，在图像处理中，通常采用 Hough 变换实现直线检测的处理，Radon 变换与 Hough 变换很相似。下面举一个利用 Radon 变换实现直线检测的例子，读者可以从中学到 Radon 变换的应用。具体方法如下：

（1）用 edge 函数提取如图 4-23 所示图像的边缘，得到二值图像，如图 4-24 所示，实现代码如下：

```
I = fitsread('solarspectra.fts');%读取原始图像
I = mat2gray(I);%将原始图像转换成灰度图，以适合于 edge 函数
BW = edge(I);%提取边缘，形成二值图像
imshow(I), figure, imshow(BW)
```

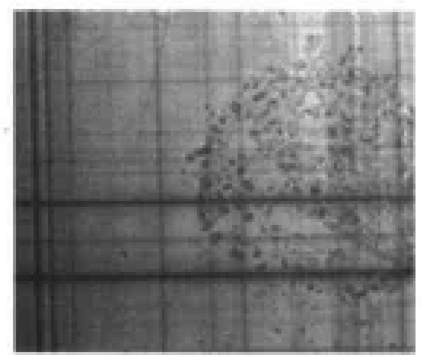


图 4-23　原始图像

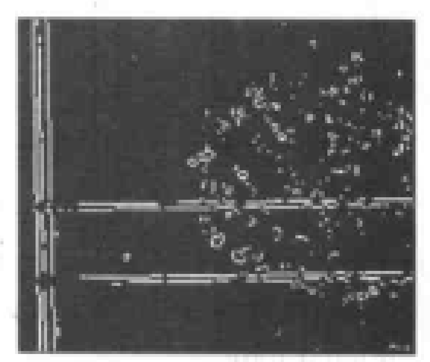


图 4-24　边缘图像

（2）对二值图像 BW 进行 Radon 变换。变换结果如图 4-25 所示。

```
theta = 0:179;
[R,xp] = radon(BW,theta);
figure, imagesc(theta, xp, R); colormap(hot);
xlabel('\theta (degrees)'); ylabel('x\prime');
```

title('R_{\theta} (x\prime)');

colorbar

（3）计算出 Radon 变换矩阵中的峰值，这些峰值对应于原始图像中的直线。如图 4-26 所示，这个例子中，R 的最强值出现在θ=1°，x'=−80 的位置，即如图 4-23 中左侧一条比较明显的暗直线。当然也可以看到对应于θ=91°处，Radon 变换也有较强的值，其对应着如图 4-23 中下部的近似水平直线。

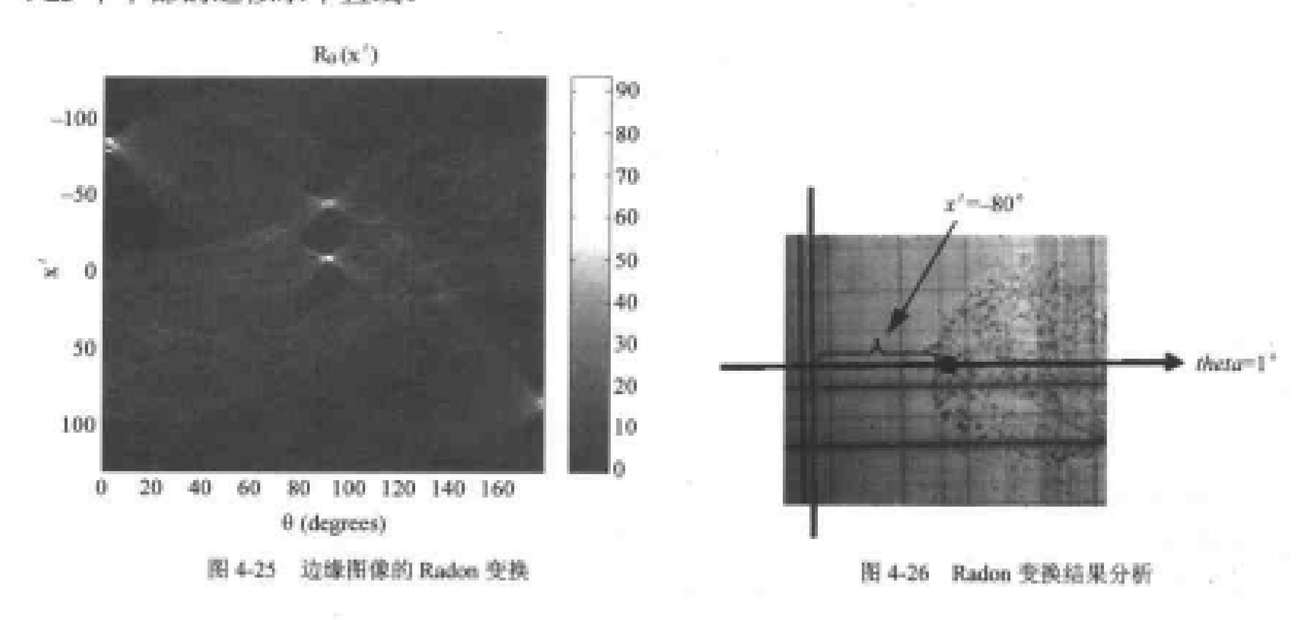


图 4-25　边缘图像的 Radon 变换

图 4-26　Radon 变换结果分析

4.3.2　逆 Radon 变换及其应用

MATLAB 工具箱提供了 iradon 函数用于实现逆 Radon 变换，并经常用于投影成像值。这个变换能把 Radon 变换反变换回来，因此可以从投影数据重建原始图像。

给定图像 I 和一系列角度 theta，radon 函数可以用来计算图像的 Radon 变换，表达式为 R=radon(R,theta)。对其计算逆 Radon 变换可以重建图像，表达式为 IR=iradon(R,theta)。

这个重建图像的例子中，投影 R 是由原始图像 I 计算得到的。然而，在大多数的应用例子中，并没有得到当前投影的原始图像。例如 X 射线吸收断层摄影术，投影是通过测量 X 射线在不同角度通过物理切片时的衰减得到的。原始图像可以认为是切片的一个截面，图像的灰度代表切片的密度。投影通过特殊的硬件设备获得，而切片内部图像通过 iradon 重建。这可以用来对活的生物体或者不透明物体实现无损成像。

iradon 函数通过平行波束的投影来重建图像。在平行波束的几何关系中，每个投影通过把特定角度穿过图像的一系列线积分组合得到。图 4-27 显示了平行波束几何在 X 射线吸收断层摄影术上的应用。注意到，在平行波束的几何关系中，有相同数目的辐射器和接收器，辐射的衰减代表物体的密度、质量等的积分，这相当于 Radon 变换中的线积分。

本图照射波束的关系是平行的，与 Radon 变换的几何关系相同。$f(x,y)$代表图像亮度，$R_\theta(x')$代表θ方向的投影。

另外一种几何关系是扇形波束，即只有一个辐射器而有多个接收器，扇形波束投影可以转换成平行波束投影，利用逆 Radon 变换来重建图像。这在下一节中还要介绍。其几何关系如图 4-28 所示。

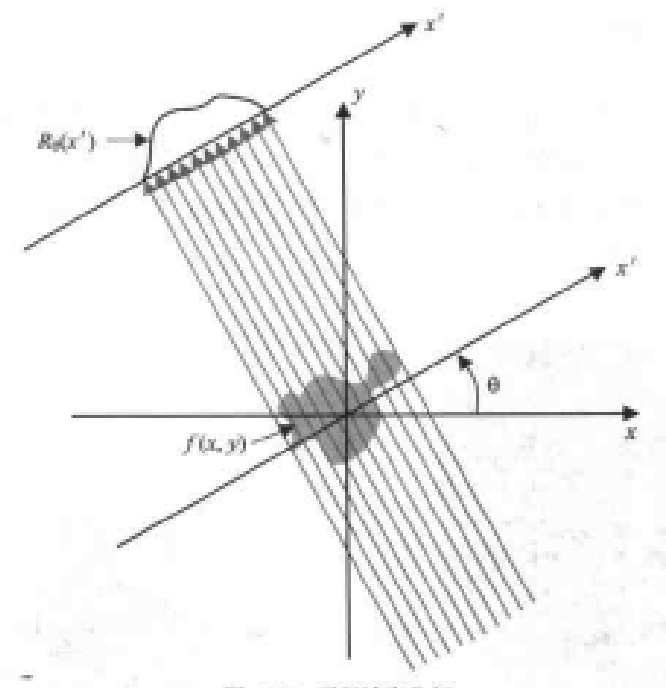


图 4-27　平行波束几何

iradon 函数利用滤波后向投影算法来计算逆 Radon 变换。这种算法利用 R 各列中的投影来构造图像 I 的近似值，若要获得更准确的图像，可以使用更多的投影值，投影数越多，重建的图像越接近原始图像。theta 矢量必须是固定增量的均匀矩阵，即每次角度的增加值$\Delta\theta$为常数。若$\Delta\theta$已知，可作为参数取代 theta 值，传入 iradon 函数。例如：

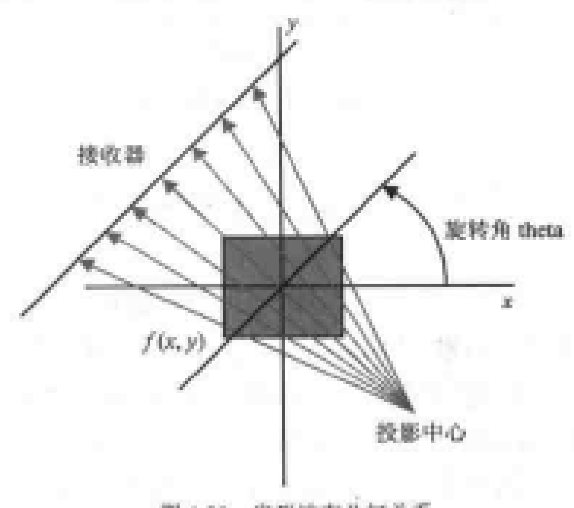


图 4-28　扇形波束几何关系

```
IR = iradon(R,Dtheta);
```

滤波 Radon 变换是首先对投影 R 滤波，再用滤波后的投影值重构图像。有些情况下，投影值含有噪声。为了消除高频噪声，可以通过加窗去除噪声。iradon 函数中可以采用多种窗函数，下面的例子采用了 hamming 窗来滤波。

```
IR = iradon(R,theta,'hamming');
```

iradon 函数也允许指定归一化频率 D，高于 D 的滤波器响应为 0，整个滤波器压缩在[0,D]范围内。这在投影时可以有效抑制高频噪声的干扰，而减小对图像重建的影响。下例调用 iradon 函数，设定归一化频率为 0.85。

```
IR = iradon(R,theta,0.85);
```

接下来介绍如何利用 radon 函数和 iradon 函数构造一个简单图像的投影并重建图像。

测试图像是 Shepp-Logan 的大脑幻影图，利用图像处理工具箱的 phantom 函数可以产生数据。Shepp-Logan 的大脑图反映了真实世界中人类大脑的很多性质。图像中外部的椭圆形是头骨，内部的椭圆是人脑的内部特征或者是肿瘤。

首先，产生如图 4-29 所示的 256 个灰度等级的大脑图，命令如下：

```
P = phantom(256);
```

```
imshow(P)
```

然后计算 3 个不同分辨率的大脑图的 Radon 变换，R1 有 18 条投影光束，R2 有 36 条投影光束，R3 有 90 条投影光束：

```
theta1 = 0:10:170; [R1,xp] = radon(P,theta1);
theta2 = 0:5:175;  [R2,xp] = radon(P,theta2);
theta3 = 0:2:178;  [R3,xp] = radon(P,theta3);
```

显示 Shepp-Logan 大脑幻影图的有 90 条投影光束的 Radon 变换图形，即 R3 的图形。结果如图 4-30 所示。

```
figure, imagesc(theta3,xp,R3); colormap(hot); colorbar
xlabel('\theta'); ylabel('x\prime');
```

利用 R1、R2 和 R3 分别进行 Shepp-Logan 大脑幻影图的重构，结果如图 4-31 所示。

```
I1 = iradon(R1,10);
I2 = iradon(R2,5);
I3 = iradon(R3,2);
imshow(I1)
figure, imshow(I2)
figure, imshow(I3)
```



图 4-29　Shepp-Logan 的大脑幻影图

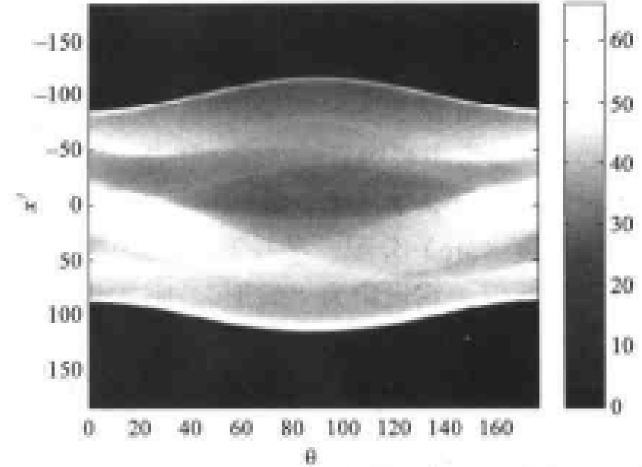
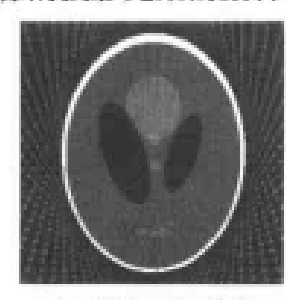


图 4-30　Shepp-Logan 的大脑图 90 条投影光束的 Radon 变换

下面用不同分辨率的逆 Radon 变换来重构图像，如图 4-31 所示。可以发现用 R1 重构图像效果最差，因为它用来重构图像的投影最少。用 R2 重构图像效果较好，用 R3 重构图像质量最好。从图中可以看出，由于用 R1 构图像投影光束太少，重构图像有很多的虚假点。为了避免这种情况，可以增加重构图像的投影光束的数目。



（a）用 R1 重构图像



（b）用 R2 重构图像



（c）用 R3 重构图像

图 4-31　重构图像

4.4 扇形光束投影

类似于 Radon 变换，扇形光束投影（Fan-Beam Projection）也是通过求沿一组特定方向的直线积分来获得图像的映射，例如对于一幅图像，就是求一组扇形直线的线积分，而这组扇形直线是由一个点源发散形成一个扇形而得名。为了获得图像的有效投影，可以改变点源的方向角，得到一组映射，如图 4-32 所示。

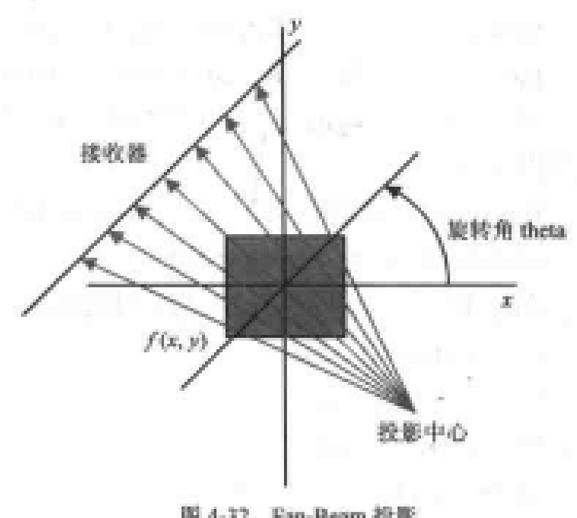


图 4-32 Fan-Beam 投影

MATLAB 提供了 fanbeam 函数来实现扇形光束投影的功能，下面分别介绍如何使用该函数获得映射数据以及从映射数据重建原图，最后给出一个例子来详细说明。

4.4.1 投影变换的基本概念

调用 fanbeam 函数实现图像映射，必须给定一些参数，比如图像、扇形光束的原点以及旋转中心（即图像中心像素），投影线的数量由 fanbeam 函数根据图像的大小以及以上给定的参数来设定。在默认情况下，其几何关系设定为“弧度”，则函数在离旋转中心距离为 D 处每隔一个弧度（arc）分配一条投影线，如图 4-33 所示。

如果将几何关系设定为“线形”，则其几何示意图如图 4-34 所示。

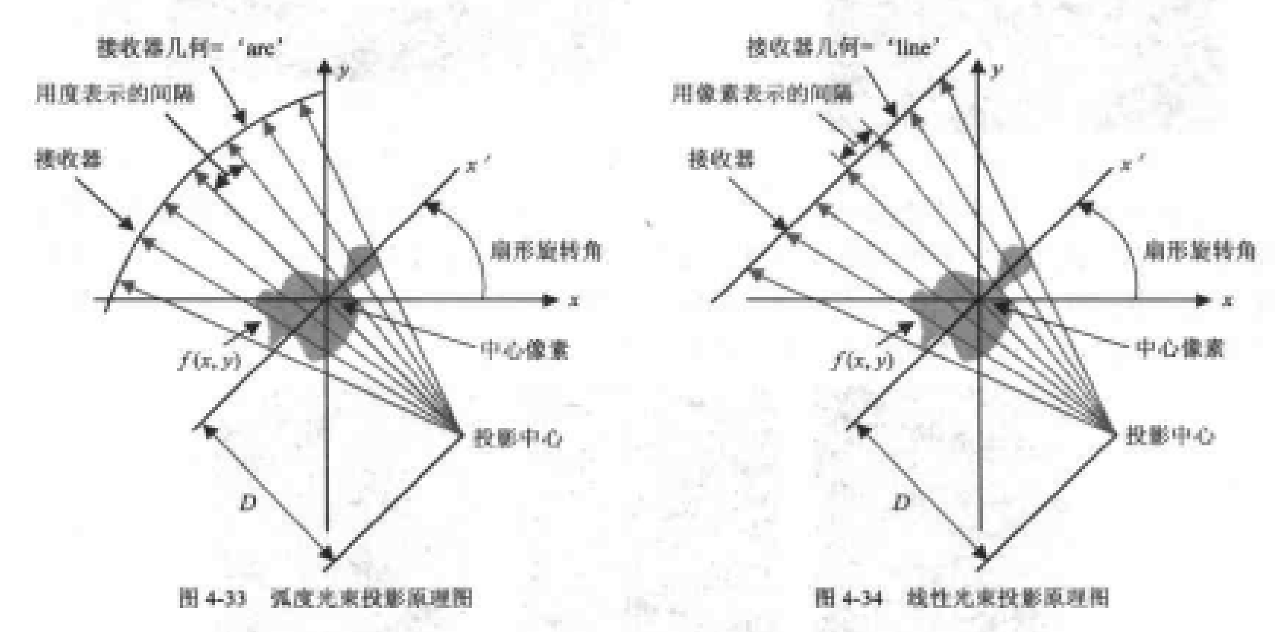


图 4-33 弧度光束投影原理图　　图 4-34 线性光束投影原理图

在获得图像的投影数据后，可以调用 ifanbeam 函数来重建图像。

I = ifanbeam(P,D);

4.4.2 投影变换函数的应用

下面以一个实例来说明映射和重建的过程。

（1）产生测试图像并显示

```
P = phantom(256);
imshow(P)
```

（2）计算投影数据

```
%设定几何关系为“线形”，分别给定光束数目为 18，36 和 90。
theta1 = 0:10:170;
[R1,xp] = radon(P,theta1);
num_angles_R1 = size(R1,2)% num_angles_R1 = 18

theta2 = 0:5:175;
[R2,xp] = radon(P,theta2);
num_angles_R2 = size(R2,2)% num_angles_R1 = 36

theta3 = 0:2:178;
[R3,xp] = radon(P,theta3);
num_angles_R3 = size(R3,2) % num_angles_R1 = 90

%设定几何关系为“弧度”，分别给定光束密度为 2 弧度、1 弧度以及 0.25 弧度。
D = 250;
dsensor1 = 2;
F1 = fanbeam(P,D,'FanSensorSpacing',dsensor1);
dsensor2 = 1;
F2 = fanbeam(P,D,'FanSensorSpacing',dsensor2);
dsensor3 = 0.25
[F3, sensor_pos3, fan_rot_angles3] = fanbeam(P,D,…
            'FanSensorSpacing',dsensor3);
```

（3）显示投影数据

```
%几何关系为“线形”
figure, imagesc(theta3,xp,R3)
colormap(hot)
colorbar
xlabel('Parallel Rotation Angle - \theta (degrees)');
ylabel('Parallel Sensor Position - x\prime (pixels)');

%几何关系为“弧度”
figure, imagesc(fan_rot_angles3, sensor_pos3, F3)
```

```
colormap(hot); colorbar
xlabel('Fan Rotation Angle (degrees)')
ylabel('Fan Sensor Position (degrees)')
```

（4）重建图像

```
%几何关系为“线形”
dtheta1 = theta1(2) - theta1(1);
I1 = iradon(R1,dtheta1,output_size);
figure, imshow(I1)

dtheta2 = theta2(2) - theta2(1);
I2 = iradon(R2,dtheta2,output_size);
figure, imshow(I2)

dtheta3 = theta3(2) - theta3(1);
I3 = iradon(R3,dtheta3,output_size);
figure, imshow(I3)

%几何关系为“弧度”
output_size = max(size(P));
Ifan1 = ifanbeam(F1,D, 'FanSensorSpacing',dsensor1,…
            'OutputSize',output_size);
figure, imshow(Ifan1)

Ifan2 = ifanbeam(F2,D, 'FanSensorSpacing',dsensor2,…
            'OutputSize',output_size);
figure, imshow(Ifan2)

Ifan3 = ifanbeam(F3,D, 'FanSensorSpacing',dsensor3,…
            'OutputSize',output_size);
figure, imshow(Ifan3)
```

图 4-35 表示由函数 phantom 产生的原始图像，图 4-36 显示了生成的投影数据，图 4-37 给出了不同投影密度重建的图像。

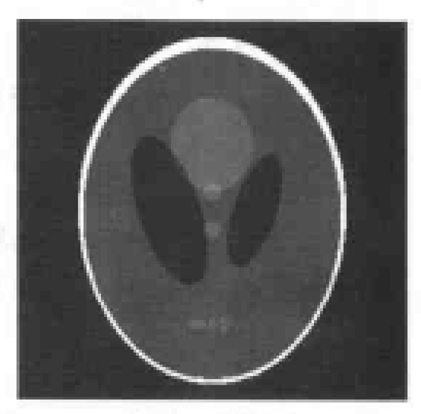
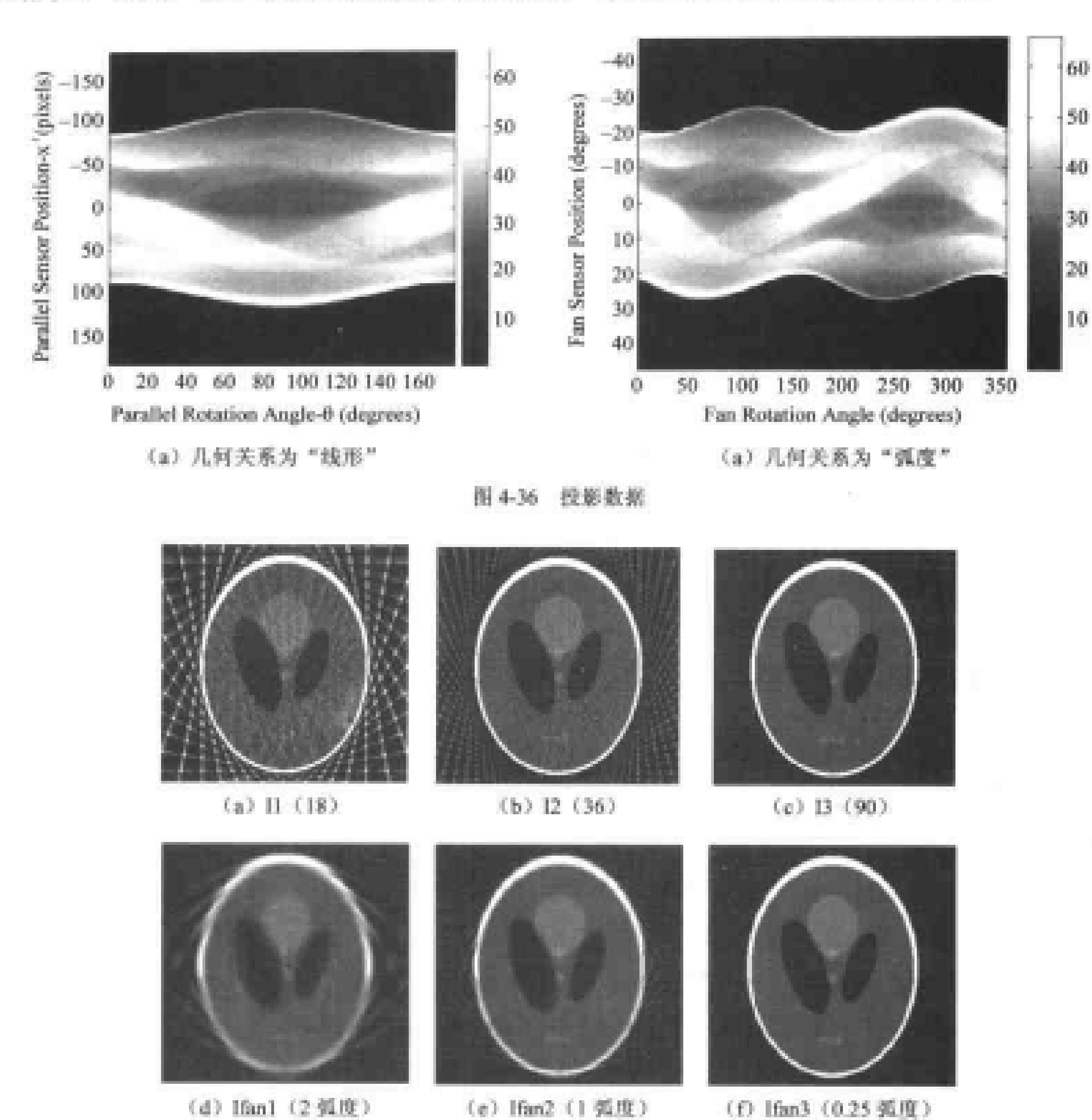


图 4-35　原始图像

比较图 4-37 的（a)、(b)、(c）3 幅图像，可以看到 I1、I2 和 I3 都不同程度存在虚假点，这三幅图像中 I3 的效果最好。比较图 4-37 的（d)、(e)、(f）3 幅图像，可以看到 Ifan1、Ifan2 和 Ifan3 都不同程度地存在模糊现象，Ifan3 的重建效果最好，而 Ifan1 模糊比较严重。重建图像中存在虚假点和模糊现象都是与用于图像重建的投影光束的数目有关，投影光束越多，重建时就能更好地体现图像的细节，而且避

免出现虚假点。显然，用 R1 和 F1 重建图像的投影光束太少，得到的重建结果明显比原始图像差很多。因此，为了获得高质量的重建结果，需要提高重建图像的投影光束。



（a）几何关系为“线形”　　（a）几何关系为“弧度”

图 4-36　投影数据

（a）I1（18）　（b）I2（36）　（c）I3（90）

（d）Ifan1（2 弧度）　（e）Ifan2（1 弧度）　（f）Ifan3（0.25 弧度）

图 4-37　依不同投影密度重建的图像

第5章　图 像 增 强

图像增强是数字图像处理的最基本的方法之一，在数字图像处理中受到广泛重视，是具有重要实用价值的技术。图像增强的目的在于:（1）采用一系列技术改善图像的视觉效果，提高图像的清晰度。(2) 将图像转换成一种更适合于人或机器进行解译和分析处理的形式。图像增强不是以图像保真度为原则，而是通过处理设法有选择地突出便于人或机器分析某些感兴趣的信息，抑制一些无用的信息，以提高图像的使用价值，即图像增强处理只是增强了对某些信息的辨别能力。

图像增强是一个相对的概念，增强效果的好坏，除与算法本身的优劣有一定的关系外，还与图像的数据特征有直接关系，同时由于评价图像质量的优劣往往凭观测者的主观而定，没有通用的定量标准，因此增强技术大多属于面向问题，增强方法只能有选择地使用。

图像增强一般包括以下内容:

- 点处理
- 空间域滤波
- 频域滤波
- 彩色增强
- 代数运算

5.1　点　处　理

点处理是通过像元亮度值（灰度值）的变换来实现的，即它将输入图像中某点（x,y）的像元值$f(x,y)$，通过映射函数 $T(\cdot)$，映射成输出图像中的像元值$g(x,y)$，即 $g(x,y)=T(f(x,y))$。根据映射方式不同，点处理可分为灰度变换和直方图调整。

5.1.1　灰度变换

灰度变换是一种简单而实用的方法。它可使图像动态范围增大，图像对比度扩展，图像变清晰，特征明显，是图像增强的重要手段之一。它可分为比例线性变换、分段线性变换和非线性灰度变换。

灰度变换是一种简单而实用的方法。它可使图像动态范围增大，图像对比度扩展，图像变清晰，特征明显。它是图像增强的重要手段之一，可分为比例线性变换、分段线性变换、非线性灰度变换

1. 比例线性变换

比例线性变换是对单波段逐个像元进行处理的，它是将原图像亮度值动态范围按线性关系式扩展到指定范围或整个动态范围。在实际运算中给定的是两个亮度区间，即要把输入图

像的某个亮度值区间$[a,b]$扩展为输出图像的亮度值区间$[a',b']$。比例线性变换对图像每一个像素灰度作线性拉伸，将有效地改善图像视觉效果。关于 a 和 b 的取值有以下两种情况，如图 5-1 所示。

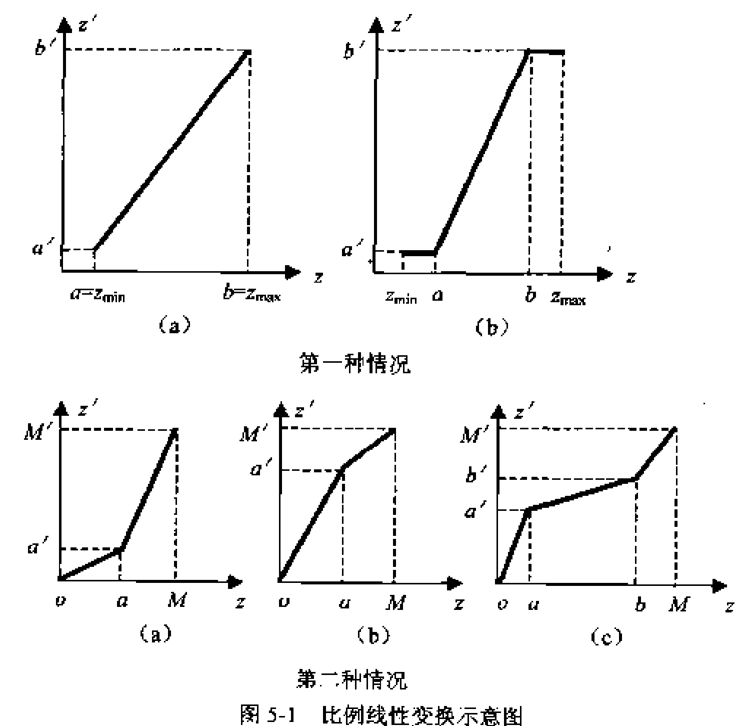


图 5-1 比例线性变换示意图

（1）原图像的最小和最大亮度值，即对原图像的灰度范围不加区别地扩展。

（2）人为规定的最小和最大值（即阈值），它把图像的低亮度值和高亮度值像元的灰度级进行了适当的归并。

2. 分段线性变换

分段线性变换是为了突出人们感兴趣的目标或亮度值区间，要求局部扩展亮度值范围。它可以有效地利用有限个灰度级，达到最大限度增强图像中有用信息的目的。

3. 非线性灰度变换

非线性灰度变换对于要进行扩展的亮度值范围是有选择的，扩展的程度是随亮度值的变化而连续变化的。有两种常用方法：

（1）对数变换，当希望对图像的低亮度区有较大的扩展而对高亮度区压缩时，可采用此种变换。

（2）指数变换，此种变换可以对图像的高亮度区给予较大的扩展。

【应用实例】

图像内各像素的灰度值是图像的重要数据，但是由于受输入或滤波等设备的影响，往往得不到图像的理想灰度值。例如在摄像机观测井下视频监控系统图像时，图像画面的感光度随场合不同会有很大差别，表现为图像的对比度不足，使图像细节不清、画面模糊。另外，对井下视频监控系统图像而言，由于环境光线不足，造成了灰度变化的不明显，为边缘特征的提取加大了难度。为此需要对图像的每一灰度级进行变换，扩大图像的灰度范围，从而增

强煤矿视频监控图像的清晰度。

（1）比例线性变换的程序设计及实现

当煤矿视频监控系统图像由于成像时感光不足或过度，或成像、记录设备的非线性动态范围太窄等因素，都会造成对比度不足的弊病，使得图像中的细节分辨不清。如图 5-2，图像中像素点的灰度全部集中在 30~140 灰度级范围内，因此，如果利用图像灰度线性扩展，可以显著改善图像的观看质量。

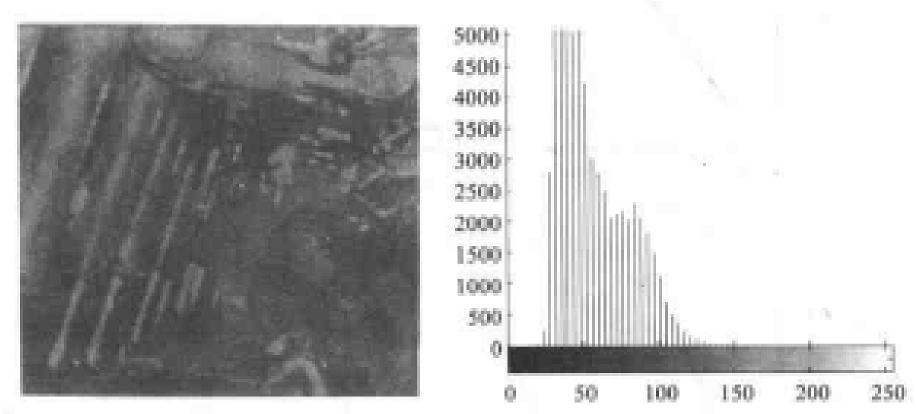


图 5-2　原始图像及其直方图

假设原图像 $f(x,y)$ 的灰度范围为 $[a,b]$，希望变换后图像 $g(x,y)$ 的动态范围为 $[c,d]$，则可以利用以下公式来实现变换，即

$$g(x,y)=\frac{(d-c)[f(x,y)-a]}{b-a}+c$$

根据图像灰度线性变换的算法，得到线性变换的 MATLAB 程序如下：

```
X1=imread('E:\temp\originalimage.tif');
figure,imshow(X1);
f0=0;g0=0;
f1=10;g1=10
f2=180;g2=1800;
f3=255;g3=255;
figure,plot([f0,f1,f2,f3],[g0,g1,g2,g3])
axistight,xlabel('f'),ylable('g')
title('intensitytransformation')
%绘制变换曲线
r1=(g1-g0)/(f1-f0);
b1=g0-r1*f0;
r2=(g2-g1)/(f2-f1);
b2=g1-r2*f1;
r3=(g3-g2)/(f3-f2);
b3=g2-r3*f2;
[m,n]=size(X1);
X2=double(X1);
for i=1:m
```

```
    for j=1:n
        f=X2(i,j);
        g(i, j)=0;
        if(f>=f1)&(f<=f2)
            g(i,j)=r1*f+b2;
        elseif(f>=f2)&(f<=f3)
            g(i,j)=r3*f+b3;
        end
    end
end
figure, imshow(mat2gray(g))
```

运行该程序后，原图像的灰度范围从原来的 30～140 扩展到 0～255，得到线性变换后的图像如图 5-3，可以看出图像质量有明显的改善。

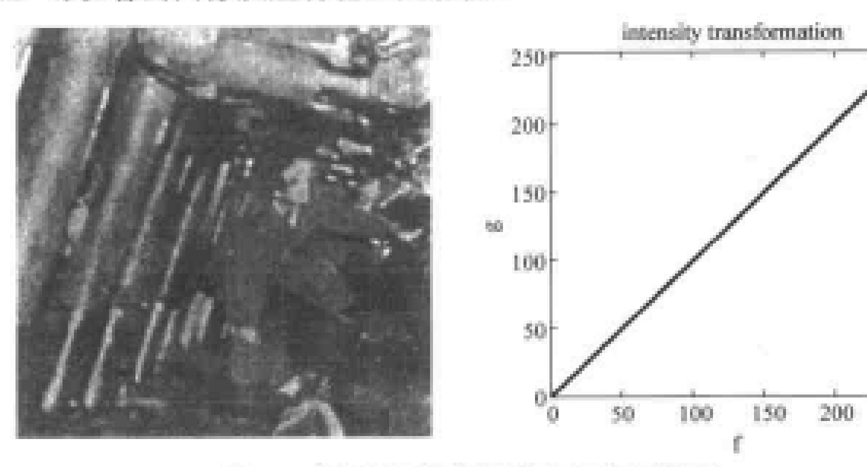


图 5-3　线性变换后图像及增强对比度的变换曲线

（2）分段线性变换的程序设计及实现

如果采用分段线性变换的方法，主要进行感兴趣区域的灰度增强，假设感兴趣区域是 (a,b)，可以采用以下的分段变换公式：

$$g(x,y)=\begin{cases} c & f(x,y)<a \\ d & f(x,y)>b \\ (d-c)[f(x,y)-a]/(b-a)+c & a<f(x,y)<b \end{cases}$$

根据图像分段灰度线性变换的算法，对感兴趣的 (a,b) 区间进行灰度变换（这里假设感兴趣区域为 20～180），可以得到分段线性变换的程序如下：

```
clear all;
X1=imread('E:\temp\originalimage.tif');
figure,
imshow(X1)
f0=0;g0=0;
f1=20;g1=10;
f2=180;g2=230;
f3=255;g3=255;
```

```
figure,plot([f0,f1,f2,f3],[g0,g1,g2,g3])
axistight,xlabel('f'),ylabel('g')
title('intensitytransformation')%绘制变换曲线
r1=(g1-g0)/(f1-f0);
b1=g0-r1*f0;
r2=(g2-g1)/(f2-f1);
b2=g1-r2*f1;
r3=(g3-g2)/(f3-f2);
b3=g2-r3*f2;
[m,n]=size(X1);
X2=double(X1);
for i=1:m
    for j=1:n
        f=X2(i,j);
        g(i,j)=0;
        if(f>=f1)&(f<=f2)
            g(i,j)=r1*f+b2;
        elseif(f>=f2)&(f<=f3)
            g(i,j)=r3*f+b3;
        end
    end
end
figure,imshow(mat2gray(g))
```

运行该程序后，得到分段线性变换后的图像如图 5-4。从图中可以看出，通过这样一个变换，原图中灰度值在 0～20 和 180～255 之间的动态范围减小了，而原图中灰度值在 180～255 之间的动态范围增加了，从而这个范围内的对比度增加了，具体表现为变换后图像中矿井支撑架亮度明显增强。

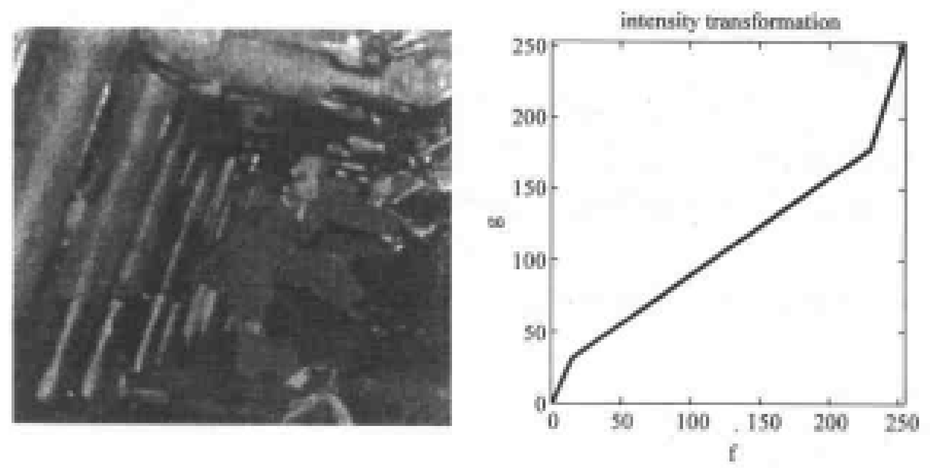


图 5-4　分段线性变换后图像及分段增强对比度的变换曲线

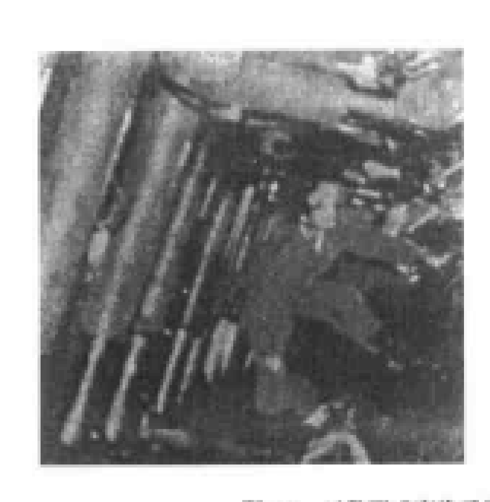
（3）非线性灰度变换的程序设计及实现（以对数变换为例）

这种方法的目标与增强对比度相反。当原图的动态范围太大，超出了某些显示设备所允

许的动态范围时，可采用对数形式的变换函数进行动态范围压缩：$g = c\log(1+f)$
其中 c 是比例尺常数，下面例程就是采用对数形式的变换函数进行动态范围压缩：

```
clear all;
X1=imread('E:\temp\originalimage.tif');
figure,
imshow(X1)
c=255/log(256);
x=0:1:255;
y=c*log(1+x);
figure,plot(x,y)
axistight,xlabel('f'),ylabel('g')
title('intensitytransformation')
%绘制变换曲线
[m,n]=size(X1);
X2=double(X1);
for i=1:m
    for j=1:n
        g(i,j)=0;
        g(i,j)=c*log(X2(i,j)+1);
    end
end
figure, imshow(mat2gray(g))
```

运行该程序后，得到对数形式变换后的图像如图 5-5。



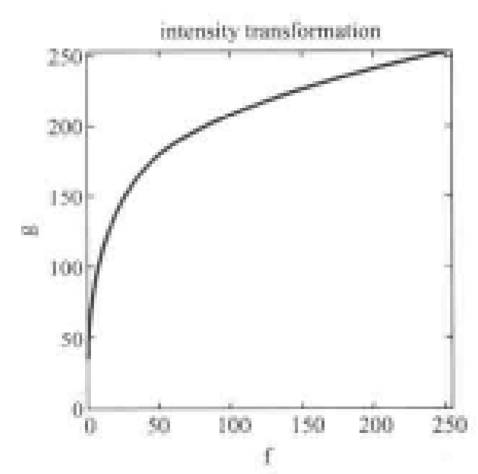


图 5-5　对数形式变换后的图像及对数形式变换曲线

从图中可以看出，通过这样一个变换，图像对比度增强了，同时突出了图像的某些细节。

5.1.2　直方图调整

一般情况下，如果图像的灰度分别集中在较窄的区间，从而引起图像细节的模糊，为了

使图像细节清晰，并使一些目标得到突出，达到增强图像的目的，可通过改善各部分亮度的比例关系，即通过直方图的方法来实现。

这种方法是以概率论为基础的，常用的方法有直方图均衡化和直方图规定化。直方图调整是以概率论为基础的。通过改变直方图的形状来达到增强图像对比度的效果，其计算方法见表 5-1 和表 5-2。常用的方法有：

1. 直方图均衡化

直方图均衡化又称直方图平坦化，是将一已知灰度概率密度分布的图像，经过某种变换，变成一幅具有均匀灰度概率密度分布的新图像，其结果是扩展了像元取值的动态范围，从而达到增强图像整体对比度的效果。

设一幅图像总像元数为 n、分 L 个灰度级，n_k 代表第 k 个灰度级 r_k 出现的频数，则第 k 灰度级出现的概率为

$$p_r(r_k)=n_k/n(0\leqslant r_k\leqslant 1,\qquad k=0,1,\cdots,L-1)$$

此时变换函数可表示为

$$s_k=T(r_k)=\sum_{j=0}^{k}p_r(r_j)=\sum_{j=0}^{k}\frac{n_j}{n}\qquad (0\leqslant r_k\leqslant 1,k=0,1,\cdots,L-1)$$

因此，根据原图像的直方图统计值就可算出均衡化后各像元的灰度值。按上式对遥感图像进行均衡化处理时，直方图上灰度分布较密的部分被拉伸；灰度分布稀疏的部分被压缩，从而使一幅图像的对比度在总体上得到很大的增强，如表 5-1 所示。

表 5-1　　直方图均衡化计算表

序号	运　算	步骤和结果							
1	列出原始图灰度 s_k（k=0,1,…,7）	0	1	2	3	4	5	6	7
2	统计原始直方图各灰度级像素 n_k	790	1023	850	656	329	245	122	81
3	同上式计算原始直方图	0.19	0.25	0.21	0.16	0.08	0.06	0.03	0.02
4	计算累积直方图	0.19	0.44	0.65	0.81	0.89	0.95	0.98	1.00
5	取整 t_k=int[$(N-1)t_k$+0.5]	1	3	5	6	6	7	7	7
6	确定映射对应关系($s_k\rightarrow t_k$)	0→1	1→3	2→5	3,4→6		5,6,7→7		
7	统计新直方图各灰度级像素 n_k		790		1023		850	985	448
8	用 $p_t(t_k)=n_k/n$		0.19		0.25		0.21	0.24	0.11

2. 直方图规定化

直方图均衡化的优点是能自动地增强整个图像的对比度，但它的具体增强效果不易控制，处理的结果总是得到全局均衡化的直方图。实际工作中有时需要变换直方图使之成为某个特定的形状，从而有选择地增强某个灰度值范围内的对比度。这时可采用比较灵活的直方图规定化方法。直方图规定化增强处理的步骤如下：

（1）如同均衡化方法中，对原始图的直方图进行灰度均衡化；

$$s_k=T(r_k)=\sum_{j=0}^{k}p_r(r_j)=\sum_{j=0}^{k}\frac{n_j}{n}\qquad (0\leqslant r_k\leqslant 1,k=0,1,\cdots,L-1)$$

（2）规定希望的密度函数，并计算能使规定的直方图均衡化的变换；

$$v_l = H(u_l) = \sum_{i=0}^{l} p_u(u_i) \qquad (0 \leqslant u_l \leqslant 1,\ l=0,1,\cdots,L-1)$$

（3）将第（1）步得到的变换反转过来，即将原始直方图对应映射到规定的直方图，也就是将所有 $p_r(r_j)$ 对应到 $p_u(u_l)$ 中去。

表 5-2　　直方图规定化计算表

序号	运　算	步骤和结果							
1	列出原始图灰度 s_k（k=0,1,…,7）	0	1	2	3	4	5	6	7
2	统计原始直方图各灰度级像素 n_k	790	1023	850	656	329	245	122	81
3	同上式计算原始直方图	0.19	0.25	0.21	0.16	0.08	0.06	0.03	0.02
4	计算累积直方图	0.19	0.44	0.65	0.81	0.89	0.95	0.98	1.00
5	规定直方图 $p(t_k)=n_k/n$	0	0	0	0.2	0	0.6	0	0.2
6	计算规定累积直方图	0	0	0	0.2	0.2	0.8	0.8	1.0
7	映射（GML）	3	5	5	5	7	7	7	7
8	查找映射对应关系	0→3	1,2,3→5			4,5,6,7→7			
9	变换后直方图	0	0	0	0.19	0	0.62	0	0.19

【应用实例】

沿用上面应用中的情况，井下监视图像明暗程度分布不均，而且灰度范围比较窄，如图 5-6 所示。

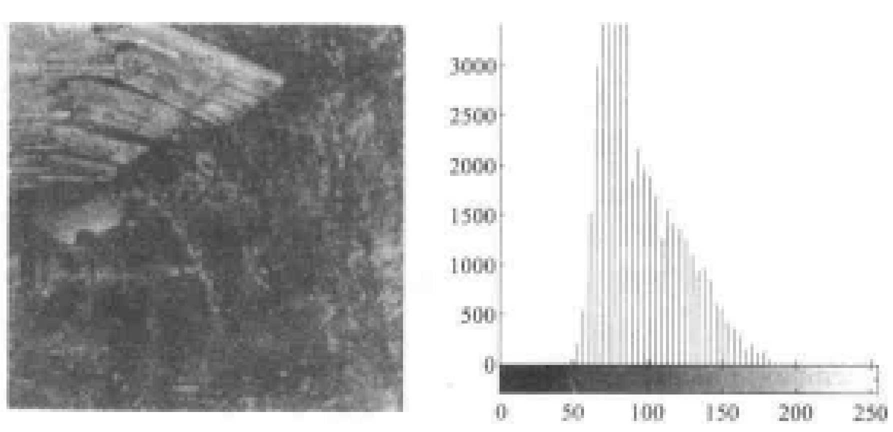


图 5-6．原始图像及其直方图统计

由图 5-6 可以看出其直方图多密集在一起且两侧较小中间突出一个高峰，这就说明图像绝大多数像素点的灰度值比较集中，使得图像信息不够丰富，图像结构不够清晰，我们可以通过直方图调整的方法进行图像增强。

（1）直方图均衡化的设计及实现

如果将直方图的高峰在水平方向压缩，向左右展开成为一个有同样高度的宽而低的新直方图，这就是图像直方图的均衡化。变换源程序如下：

```
I=imread('E:\temp\originalimage(holl).bmp');
%读取图像
J=histeq(I);
%图像直方图均衡化
figure,imshow(I)
```

```
%显示原始图像
figure,imshow(J)
%显示均衡化后图像
figure,imhist(I,64)
%对原始图像进行直方统计并显示
figure,imhist(J,64)
%对原始图像进行直方图统计并显示
```

经过均衡化的图像如图 5-7，可以看出均衡化后，原来图像中较暗区域中的一些细节更清晰。

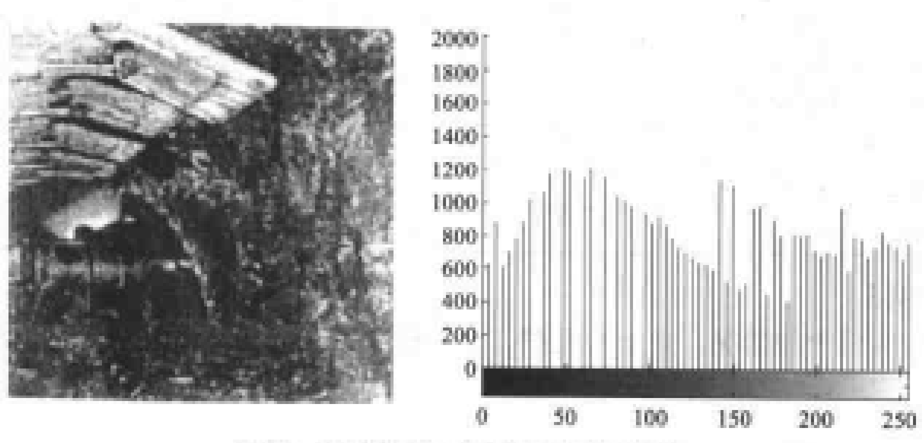


图 5-7 直方图均衡化后图像及其直方图统计

（2）直方图规定化的设计及实现

直方图均衡化对于煤矿视频监控系统图像增强非常有用，但是它只能产生一种形式的直方图，从而限制了这种方法的效能，使得它不适于交互作用的图像增强的应用。在交互处理中，常常希望根据某幅标准图像效能或抑制图像的直方图来修改原图像，有时甚至直接给出直方图的形状，希望找到某个灰度级的变换，使得原图像的直方图变成某个给定的形式，这个过程就是直方图的规定化。直方图规定化源程序如下：

```
I=imread('E:\temp\originalimage(holl).bmp');
%读取图像
hgram=0:255
%制定灰度变换范围
J=histeq(I,hgram);
%实现图像直方图规定化
figure,imshow(I)
%显示原始图像
figure,imshow(J)
%显示均衡化后图像
figure,imhist(I,64)
%对原始图像进行直方统计并显示
figure,imhist(J,64)
%对原始图像进行直方图统计并显示
```

进行直方图规定化后的图像如图 5-8 所示，可以看出，经过直方图规定化处理后的图像灰度间距拉开了，使图像变得较为清晰，图像细节也突出了。

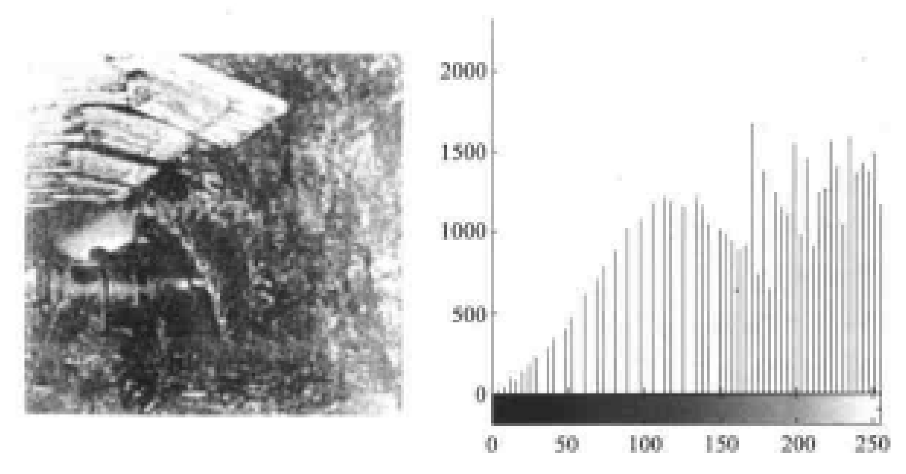


图 5-8 直方图规定化后图像及其直方图统计

5.2 空间域滤波

通常情况下，像素的邻域比该像素要大，也就是说这个像素的邻域中除了本身以外还包含了其他像素。在这种情况下，$g(x, y)$在(x, y)位置处的值不仅取决于其在该处的值，而且取决于以(x, y)为中心的邻域内所有像素的值。

为在邻域内实现增强操作，常可利用模板与图像进行卷积。每个模板实际上是一个二维数组，其中各个元素的取值确定了模板的功能，这种模板操作也称为空域滤波。

5.2.1 基本原理

根据其特点，空域滤波一般可分为线性滤波和非线性滤波两类。线性滤波器的设计常基于对傅立叶变换的分析。非线性空域滤波器则一般直接对邻域进行操作。另外各种空域滤波器根据功能又主要分成平滑滤波器和锐化滤波器。平滑可用低通来实现。平滑的目的可分为两类：一类是模糊，目的是在提取较大的目标前去除太小的细节或将目标内的小间断连接起来；另一类是消除噪声。锐化可用高通滤波来实现。锐化的目的是为了增强被模糊的细节。结合这两种分类法，可将空间滤波增强方法分成 4 类：

- 线性平滑滤波器（低通）；
- 非线性平滑滤波器（低通）；
- 线性锐化滤波器（高通）；
- 非线性锐化滤波器（高通）。

空域滤波器的工作原理都可借助频域进行分析。它们的基本特点都是让图像在傅立叶空间的某个范围的分量受到抑制，而让其他分量不受影响，从而改变输出图像的频率分布，达到增强的目的。在增强中用到的空间滤波器主要有两类：

平滑（低通）滤波器：它能减弱或消除傅立叶空间的高频分量，但不影响低频分量。因

为高频分量对应图像中的区域边缘等灰度值具有较大较快变化的部分，滤波器将这些分量滤去可使图像平滑。

锐化（高通）滤波器：它能减弱或消除傅立叶空间的低频分量，但不影响高频分量。因为低频分量对应图像中灰度值缓慢变化的区域，因而与图像的整体特性，如整体对比度和平均灰度值等有关，将这些分量滤去可使图像锐化。

空域滤波器都是利用模板卷积，主要步骤是：

（1）将模板在图中漫游，并将模板中心与图中某个像素位置重合。

（2）将模板上的系数与模板下对应的像素相乘。

（3）将所有乘积相加。

（4）将模板的输出响应赋给图中对应模板中心位置的像素。

下面重点介绍 MATLAB 中如何具体实现图像的平滑和锐化，以及此方法在工程中的应用实例。

5.2.2 平滑滤波

1. 线性平滑滤波

线性低通滤波器是最常用的线性平滑滤波器。实现这种滤波器的方法也称为邻域平均法。邻域平均法是一种局部空间域处理的算法，这种方法的基本思想是用几个像素灰度的平均值来代替每个像素的灰度。假定有一幅 $N\times N$ 个像素的图像 $f(x, y)$，平滑处理后得到一幅图像 $g(x, y)$。$g(x, y)$由下式决定：

$$g(x,y)=\frac{1}{M}\sum_{(m,n)\in S}f(m,n)$$

式中，$x,y=0,1,2,\cdots,N-1$；S 是(x,y)点邻域中点的坐标的集合，但其中不包括(x, y)点，M 是集合内坐标点的总数。上式说明，平滑后的图像 $g(x, y)$中的每个像素的灰度值均由包含在(x, y)的预定邻域中的 $f(x, y)$的几个像素的灰度值的平均值来决定。一种常见的平滑算法是将原图中一个像素的灰度值和它周围邻近八个像素的灰度值相加，然后将求得的平均值（除以9）作为新图像中该像素的灰度值。我们用如下方法来表示该操作：

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

中间的黑点表示该元素为中心元素，即该个元素是要进行处理的元素。同理可得 5×5 的模板，如下所示：

$$\frac{1}{25}\begin{bmatrix}1 & 1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1 & 1\end{bmatrix}$$

通常模板不允许移出边界，因此处理后的图像会比原图小。对于边界上无法进行模板操作的点，我们的做法是复制原图的灰度值，不再进行任何其他的处理。

另外，Wiener 滤波器也是经典的线性降噪滤波器。Wiener 滤波的思想是 20 世纪 40 年代提出来的，是一种在平稳条件下采用最小均方误差准则得出的最佳滤波准则，该方法就是寻找一个最佳的线性滤波器，使得均方误差最小。其实质是解维纳—霍夫(Wiener-Hoof)方程。

Wiener 滤波器首先估计出像素的局部矩阵均值和方差：

$$\mu=\frac{1}{MN}\sum_{n1,n2\in\eta}a(n1,n2)$$

$$\sigma^2=\frac{1}{MN}\sum_{n1,n2\in\eta}a^2(n1,n2)-\mu^2$$

η是图像中每个像素 $M\times N$ 的领域，利用 Wiener 滤波器估计出其灰度值：

$$b(n1,n2)=\mu+\frac{\sigma^2-v^2}{\sigma^2}(a(n1,n2)-\mu)$$

v^2 是整幅图像的方差，它根据图像的局部方差来调整滤波器的输出，当局部方差大时，滤波器的效果较弱，反之滤波器的效果强，是一种自适应滤波器。

【应用实例】

海上舰船目标尤其是作战舰船目标的自动识别（ATR）是国内外极为关注的热点、难点问题，也可以说是当今海上作战武器系统中的一项关键技术。图像预处理是 ATR 的重要组成部分。在图像生成和传输过程中，由于多种因素的影响，不可避免地存在噪声干扰和失真，因而总要造成图像的降质，因此在目标识别之前要进行一定的预处理，目的是为了减少图像的噪声。海天背景中的噪声主要是浪涌等引起的强脉冲性冲击噪声，可以采用邻域平均的线性平滑滤波的方法和 Wiener 滤波的方法来实现降噪处理。

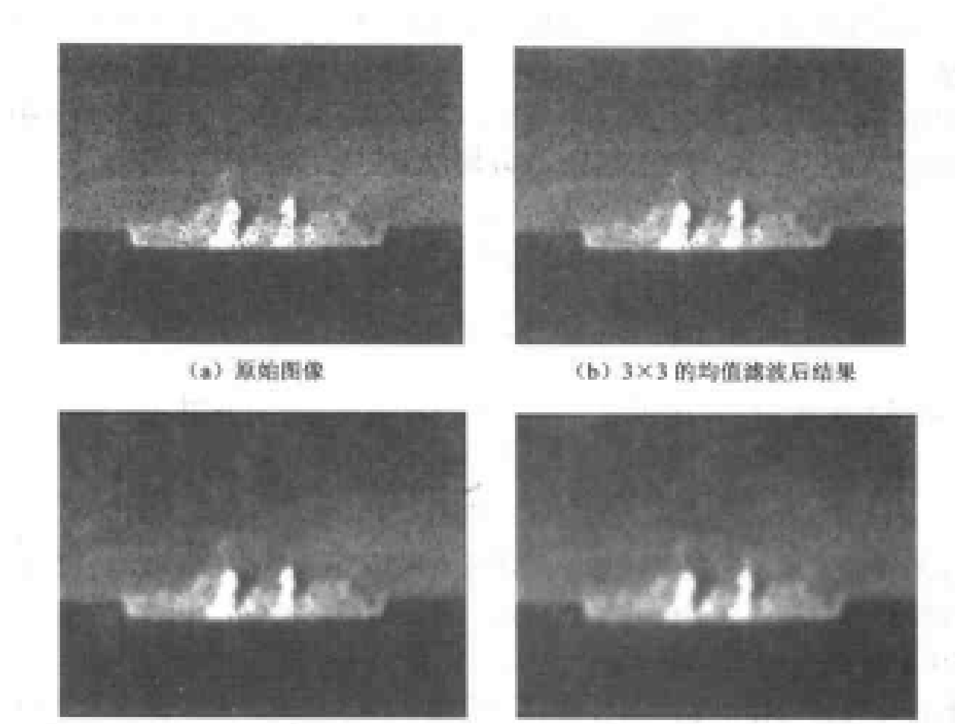
（1）邻域平均的线性平滑滤波法实现降噪的例程

```
clear all;
I=imread('E:\temp\pinghualvboyuantu.bmp');
%读入预处理图像
imshow(I)
%显示预处理图像
K1=filter2(fspecial('average',3),I)/255;
%进行 3*3 均值滤波
K2=filter2(fspecial('average',5),I)/255;
%进行 5*5 均值滤波
K3=filter2(fspecial('average',7),I)/255;
%进行 7*7 均值滤波
figure,imshow(K1)
figure,imshow(K2)
figure,imshow(K3)
```

预处理的图像和处理后的图像如图 5-9（a）～（d）所示。

比较以上采用的不同尺寸的均值滤波器仅进行低通滤波处理的结果可知，当所用的平滑

模板的尺寸增大时，消除噪声的效果增强，但同时所得的图像变得更模糊，细节的锐化程度逐步减弱。



（a）原始图像　　（b）3×3 的均值滤波后结果

（c）5×5 的均值滤波后结果　　（d）7×7 的均值滤波后结果

图 5-9　邻域线性平滑滤波

（2）Wiener 滤波法实现降噪的例程

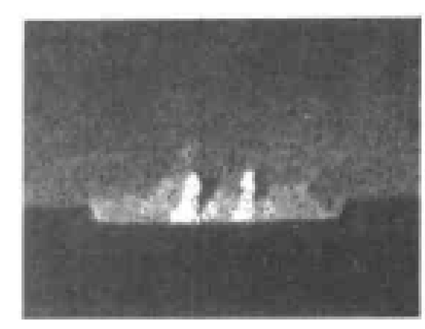
```
clear all;
I=imread('E:\temp\pinghualvboyuantu.bmp');
%原始图像读入
imshow(I)
%原始图像显示
K1=wiener2(I,[3,3]);
%3×3winner 滤波
K2=wiener2(I,[5,5]);
%5×5winner 滤波
K3=wiener2(I,[7,7]);
%7×7winner 滤波
figure,imshow(K1)
figure,imshow(K2)
figure,imshow(K3)
```

预处理的图像和处理后的图像如图 5-10（a）～（d）所示。

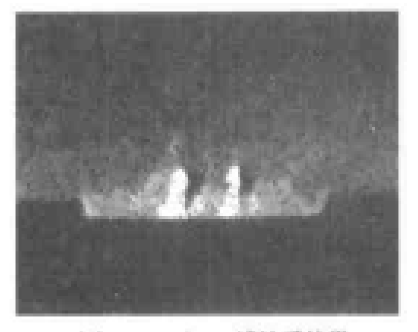
从处理后的图像看，比较邻域平均法处理后的图像，可以看出邻域平均法处理后的图像效果较差。噪声减少不明显，而且使图像的模糊度增加。相比之下 Wiener 滤波处理的图像效果比较好，图像轮廓清晰，噪声大大降低，图像质量明显提高，识别目标方便。Wiener 滤波不仅较好地消除了强脉冲性噪声的影响，而且较好地保留了图像的边缘。



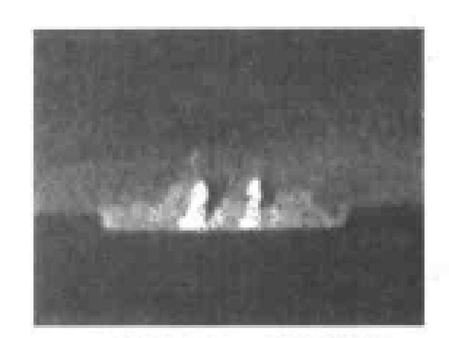
（a）原始图像



（b）B3×3Wiener 滤波后结果



（c）5×5Wiener 滤波后结果



（d）7×7Wiener 滤波后结果

图 5-10　Wiener 滤波法实现降噪

2. 非线性平滑滤波器

中值滤波是一种去除噪声的非线性处理方法，是由 Turky 在 1971 年提出的。基本原理是把数字图像或数字序列中一点的值用该点的一个邻域中各点值的中值代替。中值的定义如下：一组数 $x_1,x_2,x_3,\dots,x_n$，把 n 个数按值的大小顺序排列于下：$x_{i1}\leqslant x_{i2}\leqslant x_{i3}\leqslant\cdots\leqslant x_{in}$

$$y=Med\{x_1,x_2,x_3,\dots,x_n\}=\begin{cases}x_{i(\frac{n+1}{2})} & n\text{为奇数}\\ \frac{1}{2}\left[x_{i(\frac{n}{2})}+x_{i(\frac{n}{2}+1)}\right] & n\text{为偶数}\end{cases}$$

y 称为序列 $x_1,x_2,x_3,\dots,x_n$ 的中值。把一个点的特定长度或形状的邻域称作窗口。在一维情形下，中值滤波器是一个含有奇数个像素的滑动窗口，窗口正中间那个像素的值用窗口内各像素值的中值代替。设输入序列为 $\{x_i,\ i\in I\}$，I 为自然数集合或子集，窗口长度为 n，则滤波器输出为：

$$y_i=Med\{x_i\}=Med\{x_{i-u},\dots,x_i,\dots,x_{i+u}\}$$

其中，$i\in I$，$u=(n-1)/2$。

中值滤波的概念很容易推广到二维，此时可以利用某种形式的二维窗口。设 $\{x_{ij},(i,j)\in I^2\}$ 表示数字图像各点的灰度值，滤波窗口为 A 的二维中值滤波可定义为：

$$y_{ij}=Med_A\{x_{ij}\}=Med\{x_{i+r,j+s},(r,s)\in A,(i,j)\in I^2\}$$

二维中值滤波可以取方形，也可以取近似圆形或十字形。

中值滤波是非线性运算，因此对于随机性质的噪声输入，数学分析是相当复杂的。由大量实验可得，对于零均值正态分布的噪声输入，中值滤波输出与输入噪声的密度分布有关，输出噪声方差与输入噪声密度函数的平方成反比。

对随机噪声的抑制能力，中值滤波性能要比平均值滤波差些。但对于脉冲干扰来讲，特别是脉冲宽度较小，相距较远的窄脉冲，中值滤波是很有效的。

【应用实例】

沿用上述实例，实际应用中调用 medfilt2(A,[m,n])来实现中值滤波，例程如下：

```
clear all;
I=imread('E:\temp\pinghualvboyuantu.bmp');
%原图像读入
imshow(I)
%原始图像显示
K1=medfilt2(I,[3,3]);
%使用 3*3 模板完成中值滤波
K2=medfilt2(I,[5,5]);
%使用 5*5 模板完成中值滤波
K3=medfilt2(I,[7,7]);
%使用 7*7 模板完成中值滤波
figure,imshow(K1)
figure,imshow(K2)
figure,imshow(K3)
```

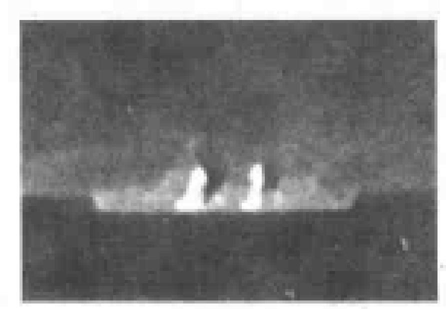
预处理的图像和处理后的图像如图 5-11（a）～（d）所示。



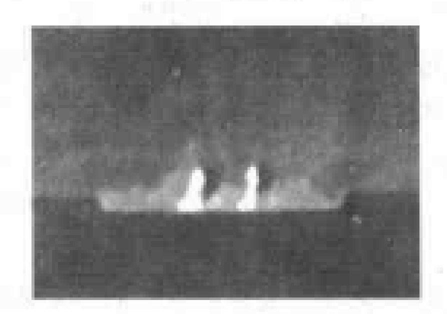
（a）原始图像



（b）B3×3 中值滤波后结果



（c）5×5 中值滤波后结果



（d）7×7 中值滤波后结果

图 5-11　调用函数 medfilt2 实现中值滤波

由以上处理结果可以看出，中值滤波器不像均值滤波器那样，它在衰减噪声的同时不会使图像的边界模糊，这也是中值滤波器受欢迎的主要原因。中值滤波器去噪声的效果依赖于两个要素：邻域的空间范围，中值计算中所涉及的像素数。一般来说，小于中值滤波器面积一半的亮或暗的物体基本上会被滤掉，而较大的物体则几乎会原封不动地保存下来。因此中值滤波器的空间尺寸必须根据手中的问题来进行调整。较简单的模板是 $N\times N$ 的方形（这里 N 通常是奇

数），计算时用到所有 N 个像素点。另外，我们也可以使用稀疏分布的模板来节省时间。

常用的稀疏距阵有：

$$domain=\begin{bmatrix}0&0&1&0&0\\0&0&1&0&0\\1&1&1&1&1\\0&0&1&0&0\\0&0&1&0&0\end{bmatrix}\quad domain=\begin{bmatrix}1&0&0&0&1\\0&1&0&1&0\\0&0&1&0&0\\0&1&0&1&0\\1&0&0&0&1\end{bmatrix}$$

这里仅举两例，例中都是对于 5×5 的模板来说的。当然，实际应用中可以根据不同的情况选取不同大小的模板，来达到更好的应用效果。

需要说明的是，中值滤波只是排序统计滤波中的一种，即用当前窗中灰度排序在中间的值代替当前点的值。我们在实现中也可以用其他规定点的值代替，在 MATLAB 中这种滤波器可以用 ordfilt2(A,order,domain)函数来实现，分别举例如下：

B=ordfilt2(A,1,ones(3,3))实现 3×3 的最小值滤波器，因为它取全 1 模板中排在最小位置处的那个像素。

B=ordfilt2(A,9,ones(3,3))实现 3×3 的最大值滤波器，因为它取全 9 模板中排在最大位置处的那个像素。

B=ordfilt2(A,1,[0 1 0；1 0 1；0 1 0])的输出是每个像素的东、西、南、北四个方向相邻像素灰度的最小值，因为它取四相邻的模板中排在最小位置处的那个像素。

5.2.3 锐化滤波

在图像识别中，需要有边缘鲜明的图像，即图像锐化。图像锐化的目的是为了突出图像的边缘信息，加强图像的轮廓特征，以便于人眼的观察和机器的识别。因此，从增强的目的看，它是与图像平滑相反的一类处理。

图像中对象的边缘像素都是两部变化较大的地方。而边缘模糊、线条不均是由于减少了边缘亮度差异的缘故。从数学观点来看，检查图像某区域内灰度的变化就是微分的概念，因此可以通过微分的方法进行图像锐化。根据微分方法是否线性，可将锐化分为线性锐化和非线性锐化两类，下面分别介绍。

1. 线性锐化滤波

线性高通滤波器是最常用的线性锐化滤波器，这种滤波器的中心系数都是正的，而周围的系数都是负的。对 3×3 的模板来说，典型的系数取值是：

[0 −1 0；−1 4 −1；0 −1 0]

事实上这是拉普拉斯算子。

拉普拉斯算子是实线性导数运算，对被运算的图像它满足各向同性的要求，这对于图像增强是非常有利的。拉氏算子的表达式是

$$\nabla^2=\frac{\partial^2 f}{\partial x^2}+\frac{\partial^2 f}{\partial y^2}$$

对于离散函数 $f(i,j)$，其差分形式是：

$$\nabla^2 f(i,j) = \Delta x^2 f(i,j) + \Delta y^2 f(i,j)$$

这里 $\Delta x^2 f(i,j)$ 和 $\Delta y^2 f(i,j)$ 是 $f(i,j)$ 在 x 方向和 y 方向的二阶差分，所以离散函数的拉氏算子的表达式为：

$\nabla^2 f(i,j) = f(i+1,j) + f(i-1,j) + f(i,j+1) + f(i,j-1) - 4f(i,j)$ 系数取值是：

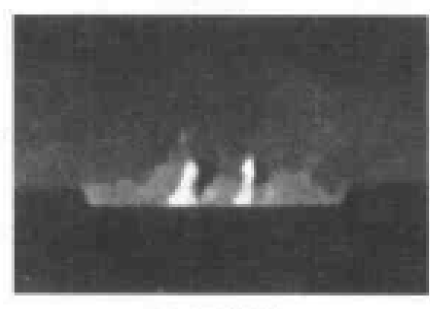
[0 −1 0；−1 8 −1；0 −1 0]

如上所述。

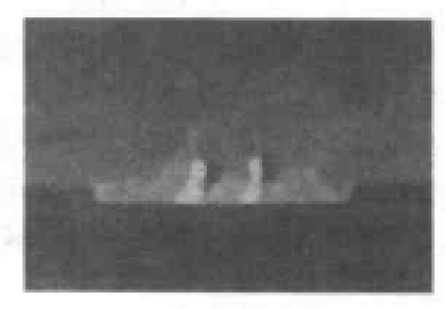
在 MATLAB 中可通过调用 filter2 函数和 fspecial 函数来实现，代码如下：

```
clear all;
I1=imread('E:\temp\ruihuatuxiang.bmp');
%图像读入
I1=double(I1);
%将图像距阵转化为 double 型
h1=fspecial('laplacian');
%拉氏变换
I2=filter2(h1,I1);
figure,imshow(I1,[]);
I3=I1-I2;
%增强图像为原图像减去拉氏变换后的结果
figure,imshow(I3,[]);
```

处理前后的图像分别如图 5-12 所示。



（a）原始图像



（b）拉氏增强后图像

图 5-12 拉普拉斯图像增强

关于线性锐化滤波后面的边缘检测中还将详细介绍，这里只做简单介绍。

2. 非线性锐化滤波

对一幅图像施加梯度模算子，可以增强灰度变化的幅度，因此我们可以采用梯度模算子作为图像的锐化算子。此方法也是最常用的非线性锐化滤波方法，而且由数学知识我们知道，梯度模算子具有方向同性和位移不变性，这正是我们所希望的。

对于离散函数 $f(i,j)$，利用差分来代替微分。

一阶差分的定义为：

$$\Delta_x f(i,j) = f(i,j) - f(i-1,j)$$

$$\Delta_y f(i,j) = f(i,j) - f(i,j-1)$$

因此，梯度的定义为：

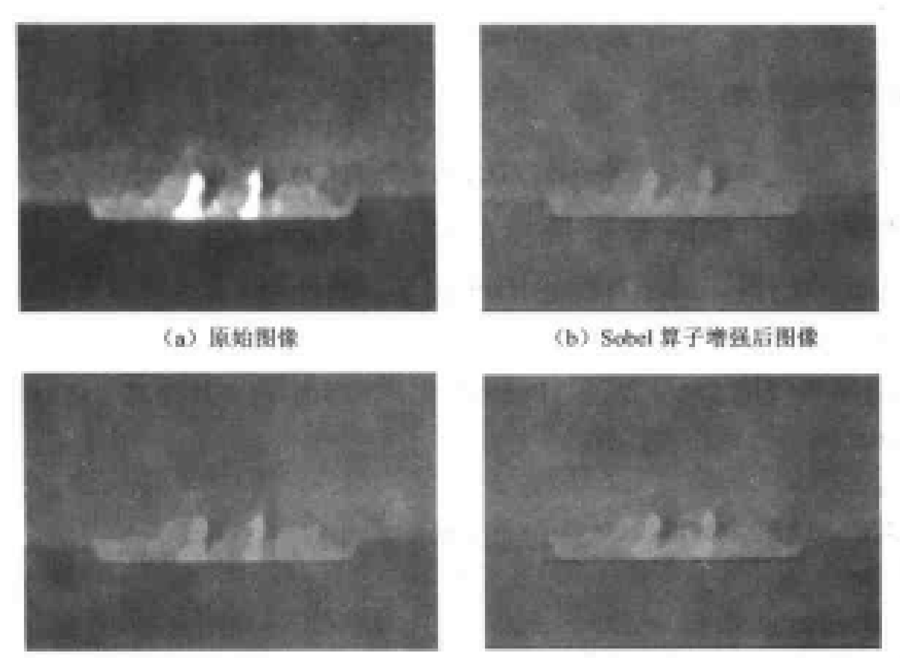
$$|G| = \left[\Delta_x f(i,j)^2 - \Delta_y f(i,j)^2\right]^{\frac{1}{2}}$$

为了运算简便，实际中采用梯度模的近似形式，如 $|\Delta_x f(i,j)| + |\Delta_x f(i,j)|$ 、$\max(|\Delta_x f(i,j)|, |\Delta_y f(i,j)|)$ 、$\max|f(i,j) - f(m,n)|$ 等。另外，还有一些常用的算子，如 Roberts 算子和 Sobel 算子。

以下例程分别实现用 Sobel 算子、Prewitt 算子和高斯-拉普拉斯算子进行图像锐化。

```
%应用 Sobel 算子进行图像锐化
clear all;
I1=imread('E:\MATLAB7\work\5-2\yuantu.bmp');
I1=double(I1);
h1=fspecial('sobel');
I2=filter2(h1,I1);
figure,imshow(I1,[]);
I3=I1-I2;
%%增强图像为原图像减去 Sobel 算子运算后的结果
figure,imshow(I3,[]);
%应用 Prewitt 算子进行图像锐化
clear all;
I1=imread('E:\MATLAB7\work\5-2\yuantu.bmp');
I1=double(I1);
h1=fspecial('prewitt');
I2=filter2(h1,I1);
figure,imshow(I1,[]);
I3=I1-I2;
%%增强图像为原图像减去 Prewitt 算子运算后的结果
figure,imshow(I3,[]);
%应用高斯-拉普拉斯算子进行图像锐化
clear all;
I1=imread('E:\MATLAB7\work\5-2\yuantu.bmp');
I1=double(I1);
h1=fspecial('log');
I2=filter2(h1,I1);
figure,imshow(I1,[]);
I3=I1-I2;
%%增强图像为原图像减去高斯-拉普拉斯算子运算后的结果
figure,imshow(I3,[]);
```

处理前后的图像分别如图 5-13 所示。



（a）原始图像　　（b）Sobel 算子增强后图像

（c）Prewitt 算子增强后图像　　（d）高斯-拉普拉斯算子增强后图像

图 5-13　图像增强效果对比

容易看出经过这些算子进行运算后，图像得到了明显的增强。

5.3 频域滤波

频域滤波的基础是傅立叶变换和卷积定理，即 $G(u,v)=H(u,v)F(u,v)$，式中 $G(u,v)$ 为增强后的图像，$H(u,v)$ 为传递函数，$F(u,v)$ 为待增强的图像。一般的频域滤波可以分为以下几类：

- 低通滤波
- 高通滤波
- 带通滤波
- 同态滤波

5.3.1 低通滤波

在频率域中，通过滤波器函数衰减高频信息而使低频信息畅通无阻的过程称为低通滤波。低通滤波抑制了反映灰度聚变边界特征的高频信息以及包括在高频中的孤立点噪声，起到平滑图像去噪声的增强作用。下面介绍常用的几种滤波器函数，由于它们以完全相同的方式修改图像频谱的实部和虚部，而不改变频谱的相位，因而被称为零相位移滤波器函数。

1. 理想滤波器

二维理想低通滤波器（ILPF）函数如下：

$$H(u,v)=\begin{cases}1 & D(u,v)>D_0\\ 0 & D(u,v)\leqslant D_0\end{cases}$$

式中 D_0 是一个规定的非负的量，$D(u,v)$是从点(u,v)到频率平面的原点的距离，即

$D(u,v)=\sqrt{u^2+v^2}$。理想低通滤波器传递函数的透视图和其剖面图如图 5-14 所示。对于理想低通滤波器的剖面，在 $H(u,v)=1$ 和 $H(u,v)=0$ 之间的跳跃点（D_0）通常称为截止频率。

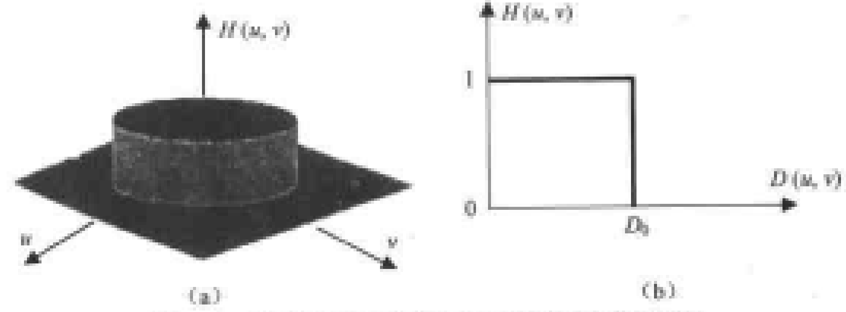


图 5-14　理想低通滤波器传递函数的透视图和其剖面图

2. 巴特沃斯滤波器

n 阶巴特沃斯（Butterworth）低通滤波器（BLPF）函数由下式决定：

$$H(u,v)=\frac{1}{1+\left[D(u,v)/D_0\right]^{2n}}$$

巴特沃斯低通滤波器传递函数的透视图和剖面图如图 5-15 所示。它的特性为连续性衰减，而不像理想低通滤波器那样陡峭和明显的不连续性。采用该滤波器在抑制噪声的同时，图像边缘的模糊程度大大减小，没有振铃效应产生。

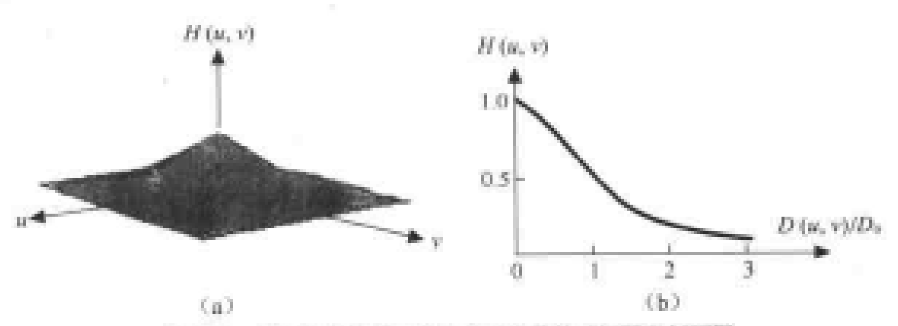


图 5-15　巴特沃斯低通滤波器传递函数的透视图和剖面图

3. 指数滤波器

指数滤波器（ELPF）函数由下式决定：

$$H(u,v)=\mathrm{e}^{-\left[\frac{D(u,v)}{D_0}\right]^n}$$。

变量 n 控制从原点算起的距离函数 $H(u,v)$的增长率。

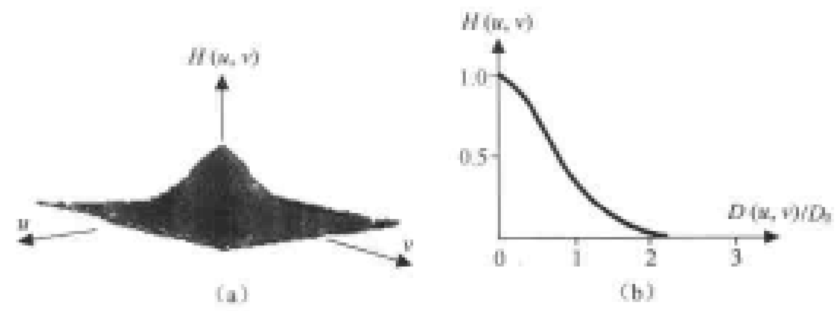


图 5-16　指数滤波器传递函数的透视图和剖面图

4. 梯形滤波器

梯形滤波器（TLPF）函数是对理想低通滤波器函数和完全平滑低通滤波器函数的折中。它的传递函数为：

$$H(u,v)=\begin{cases}1 & D(u,v)<D_0\\ [D(u,v)-D_1]/(D_0-D_1) & D_0\leqslant D(u,v)\leqslant D_1\\ 0 & D(u,v)>D_1\end{cases}$$

式中 D_0 和 D_1 是规定的。梯形滤波器的透视图和剖面图如图 5-17 所示。

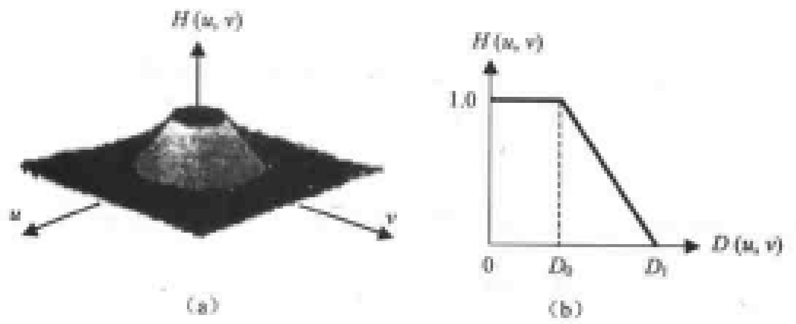


图 5-17　梯形滤波器的透视图和剖面图

【应用实例】

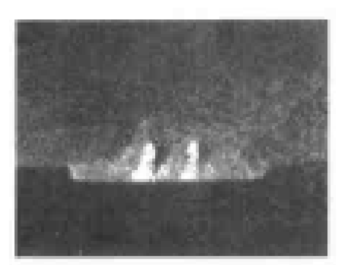
图像的能量大部分集中在幅度谱的低频和中频度，而图像的边缘和噪声对应于高频部分。在上一节所述的海上舰船目标识别中，目标图像在生成和传输过程中，由于多种因素的影响，存在噪声干扰。其中海天背景中的噪声主要是浪涌等引起的强脉冲性冲激噪声，在图像中主要分布在高频区，可以采用低通波的方法来实现降噪处理。

Buterworth 低通滤波器实现例程如下：

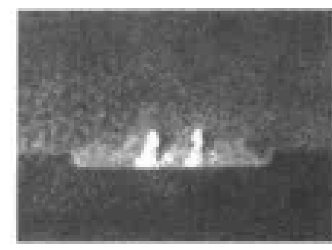
```
clear all;
I1=imread('E:\MATLAB7\work\5-2\pinyulvboyuantu.bmp');
figure,imshow(I1,[]);
f=double(I1);
g=fft2(f);
%傅立叶变换
g=fftshift(g);
%转换数据矩阵
[N1,N2]=size(g);
n=2;
d0=50;
n1=fix(N1/2);
n2=fix(N2/2);
for i=1:N1
    for j=1:N2
        d=sqrt((i-n1)^2+(j-n2)^2);
        h=1/(1+0.414*(d/d0)^(2*n));
        %计算 Butterworth 低通转换函数
```

```
            result(i,j)=h*g(i,j);
        end
    end
    result=ifftshift(result);
    X2=ifft2(result);
    X3=uint8(real(X2));
    figure,imshow(X3)
```

原图和处理结果如图 5-18 所示。



（a）原始图像



（b）滤波去噪后图像

图 5-18　Butterworth 低通滤波

比较两图可以看出处理后噪声大大减弱。

低通指数滤波和低通梯形滤波实现方法同于上述方法，只是滤波器有所改变而已。

5.3.2　高通滤波

高通滤波是为了衰减或抑制低频分量，让高频分量畅通的滤波。因为边缘及灰度急剧变化部分与高频分量相关联，在频率域中进行高通滤波将使图像得到锐化处理。与低通滤波相对应，下面介绍的滤波器也是零相位移滤波器函数。

1. 理想滤波器

二维理想高通滤波器（IHPF）函数如下：

$$H(u,v)=\begin{cases}1 & D(u,v)>D_0\\0 & D(u,v)\leqslant D_0\end{cases}$$

式中 D_0 是一个规定的非负的量，$D(u,v)$是从点(u,v)到频率平面的原点的距离，即 $D(u,v)=\sqrt{u^2+v^2}$ 。

理想高通滤波器传递函数的透视图和其剖面图如图 5-19 所示。对于理想高通滤波器的剖面，在 $H(u,v)=1$ 和 $H(u,v)=0$ 之间的跳跃点（D_0）通常称为截止频率。

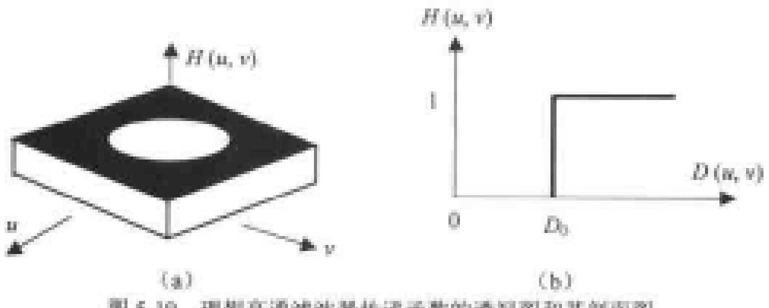


图 5-19　理想高通滤波器传递函数的透视图和其剖面图

2. 巴特沃斯滤波器

n 阶巴特沃斯（Butterworth）高通滤波器（BHPF）函数由下式决定，如图 5-20 所示。

$$H(u,v)=\frac{1}{1+[D_0/D(u,v)]^{2n}}$$

3. 指数滤波器

指数滤波器（EHPF）函数由下式决定，如图 5-21 所示。

$$H(u,v)=\mathrm{e}^{-\left[\frac{D_0}{D(u,v)}\right]^n}$$

变量 n 控制从原点算起的距离函数 $H(u,v)$ 的增长率。

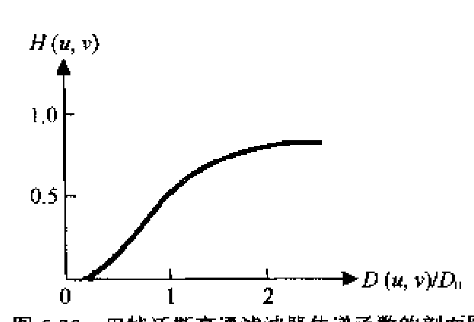


图 5-20　巴特沃斯高通滤波器传递函数的剖面图

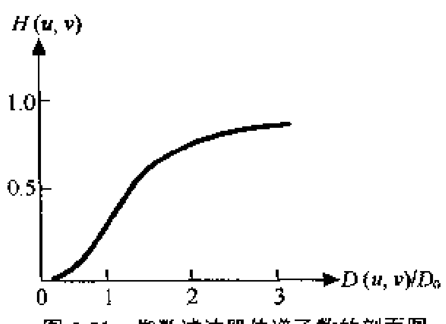


图 5-21　指数滤波器传递函数的剖面图

4. 梯形滤波器

梯形滤波器（THPF）函数是对理想低通滤波器函数和完全平滑低通滤波器函数的折中。它的传递函数为：

$$H(u,v)=\begin{cases}1 & D(u,v)>D_1\\ [D(u,v)-D_0]/(D_1-D_0) & D_0\leqslant D(u,v)\leqslant D_1\\ 0 & D(u,v)<D_0\end{cases}$$

式中 D_0 和 D_1 是规定的。梯形滤波器的透视图和剖面图如图 5-22 所示。

如果仍对上述图像进行高通滤波，例程如下：

```
clear all;
I1=imread('E:\MATLAB7\work\5-2\pinyulvboyuantu.bmp');
figure,imshow(I1,[]);
f=double(I1);
g=fft2(f);
g=fftshift(g);
[N1,N2]=size(g);
n=2;
d0=5;
n1=fix(N1/2);
n2=fix(N2/2);
for i=1:N1
```

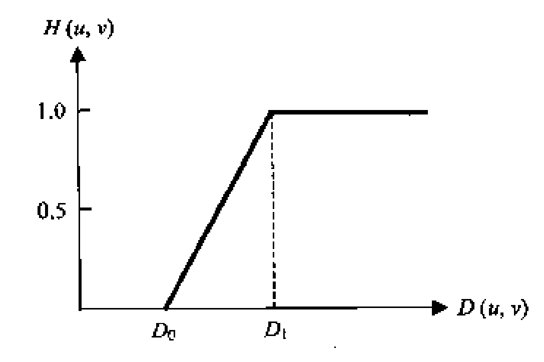
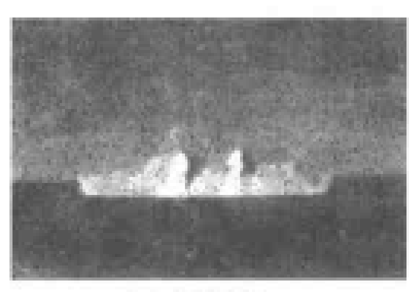


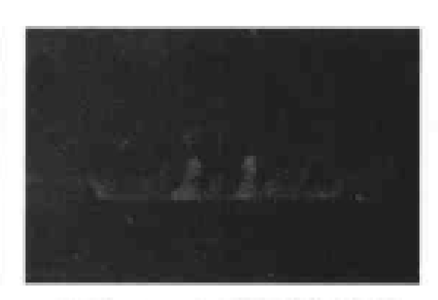
图 5-22　梯形滤波器的剖面图

```
        for j=1:N2
            d=sqrt((i-n1)^2+(j-n2)^2);
            if d==0
                h=0;
            else
                h=1/(1+(d0/d)^(2*n));
            end
            result(i,j)=h*g(i,j);
        end
    end
    result=ifftshift(result);
    X2=ifft2(result);
    X3=uint8(real(X2));
    figure,imshow(X3)
```

原图和处理结果如图 5-23 所示。



（a）原始图像



（b）Butterworth 高通滤波去噪图像

图 5-23 Butterworth 高通滤波

由图 5-23 可以看出，图像比较昏暗，很多细节都看不清了。这是因为如前所述，图像的大部分能量集中在低频区域，经过高通滤波后，虽然各区域边界得到了增强，但是图像中的低频部分被滤除，原图的灰度动态范围被压缩，所以图像比较昏暗。

同于低频滤波的情况，高通指数滤波和高通梯形滤波实现方法与上述高通 Butterworth 滤波方法相同，只是滤波器有所改变而已。

5.3.3 带通滤波器

带通滤波器允许一定频率范围内的信号通过而阻止其他频率范围内的信号通过。通常使用的带通滤波器函数为高斯滤波器函数，它的表达式如下：

$$H(u,v)=A\exp\left[-\frac{D^2(u,v)}{2a_1^2}\right]-B\exp\left[-\frac{D^2(u,v)}{2a_{21}^2}\right]$$

式中 A、B 为可选择的参数，$D(u,v)$、a_1 和 a_2 是共同决定高斯函数形态的参数。由式可见函数是由两个宽度不同的高斯函数叠加合成的。由于在频率域的高频和低频部分均有衰减，所以称之为带通滤波，如图 5-24 所示。

以上介绍了各种低通、高通和带通滤波器，究竟选择哪种滤波器交换，选择什么参数合

适，这要根据图像处理时需要增强的内容来确定。

此方法可以根据需要，对图像中感兴趣的频率分量进行处理，方法和低通以及高通滤波方法相同，只是选择的模板不同而已，例如可以选为

$$domain=\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

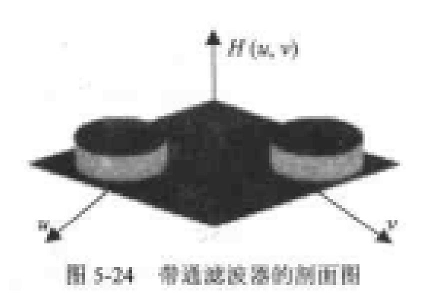


图 5-24 带通滤波器的剖面图

5.3.4 同态滤波增强

同态滤波增强是把频率过滤和灰度变换结合起来的一种处理方法。它是把图像的照明反射模型作为频域处理的基础，利用压缩灰度范围和增强对比度来改善图像的一种处理技术。它在密度域中运用相当成功。

一幅图像 $f(x,y)$ 可以看成由两个分量组合而成，即

$$f(x,y)=i(x,y)\cdot(x,y)$$

$i(x,y)$ 为照明分量（入射分量），是入射到景物上的光强度；

$r(x,y)$ 为反射分量，是受到景物反射的光强度。

具体步骤如下：

（1）先对上式的两边同时取对数，即

$$\ln[f(x,y)]=\ln[i(x,y)]+\ln[r(x,y)]$$

（2）将上式两边取傅立叶变换，得

$$F(u,v)=I(u,v)+R(u,v)$$

（3）用一个频域函数 $H(u,v)$ 处理 $F(u,v)$，可得到

$$H(u,v)F(u,v)=H(u,v)I(u,v)+H(u,v)R(u,v)$$

（4）逆傅立叶变换到空间域得

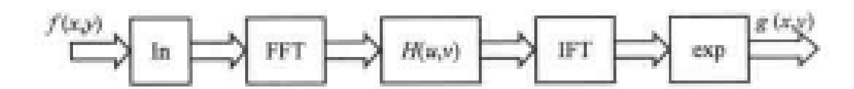
$$h_f(x,y)=h_i(x,y)+h_r(x,y)$$

可见增强后得图像是由对应照度分量与反射分量两部分叠加而成。

（5）再将上式两边取指数，得

$$g(x,y)=\exp(|h_f(x,y)|)=\exp(|h_i(x,y)|)+\exp(|h_r(x,y)|)$$

这里，称作同态滤波函数，它可以分别作用于照度分量和反射分量上。



一幅图像的照明分量通常用慢变化来表征，而反射分量则倾向于急剧变换，所以图像取对数后傅立叶变换的低频部分主要对应照度分量，而高频部分主要对应反射分量。适当的选择滤波器函数将会对傅立叶变换中的低频部分和高频部分产生不同的响应。处理结果会使像元灰度的动态范围或图像对比度得到增强。

5.4 彩 色 增 强

彩色增强在图像处理中应用十分广泛且效果显著。人的视觉系统对彩色相当敏感，人眼一般能区分的灰度级只有二十多个，而对不同亮度和色调的彩色图像分辨能力可达到灰度分辨能力的百倍以上。彩色增强就是根据人的这个特点，将彩色用于图像增强之中，从而提高了图像的可分辨性。常见的彩色增强技术有以下几类：

- 真彩色增强技术
- 伪彩色增强技术
- 假彩色增强技术
- HIS 变换增强

5.4.1 真彩色增强技术

一般把能真实反应或近似反应地物本来颜色的图像叫真彩色图像。例如 TM 图像的三个可见光波段 TM1（蓝光波段）、TM2(绿光波段)和 TM3(红光波段)的合成图像就近似真彩色图像，另外真彩色航空照片也是真彩色图像。

5.4.2 伪彩色增强技术

伪彩色增强是将一个波段或单一的黑白图像变换为彩色图像，从而把人眼不能区分的微小的灰度差别显示为明显的色彩差异，更便于解译和提取有用信息。

伪彩色增强的方法主要有以下三种：

1. 密度分割法

密度分割或密度分层是伪彩色增强中最简单的一种方法，它是对图像亮度范围进行分割，使一定亮度间隔对应于某一类地物或几类地物从而有利于图像的增强和分类。如图 5-25 所示，它是把黑白图像的灰度级从 0（黑）到 M0（白）分成 N 个区间 L_i，i=1,2,…,N。给每个区间 L_i 指定一种彩色 C_i，这样，便可以把一幅灰度图像变成一幅伪彩色图像。此法比较直观简单，缺点是变换出的彩色数目有限。

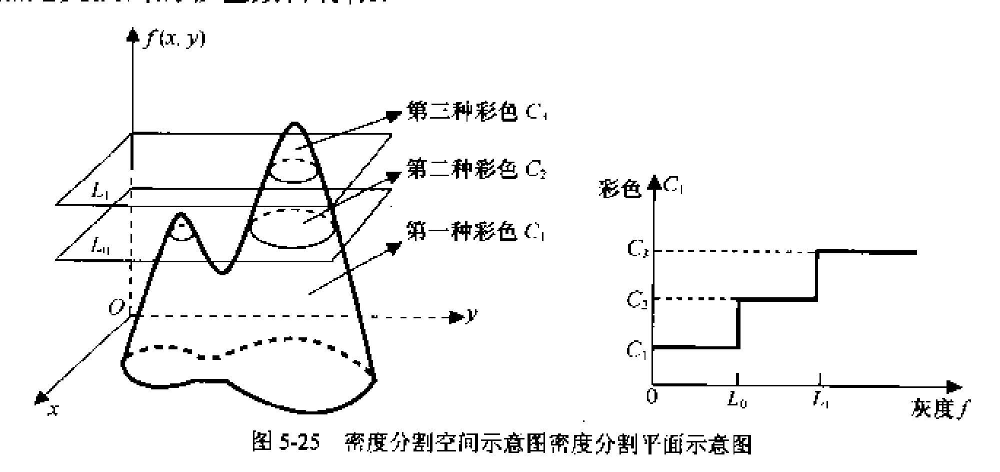
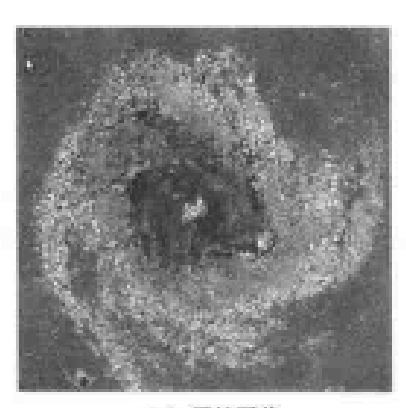


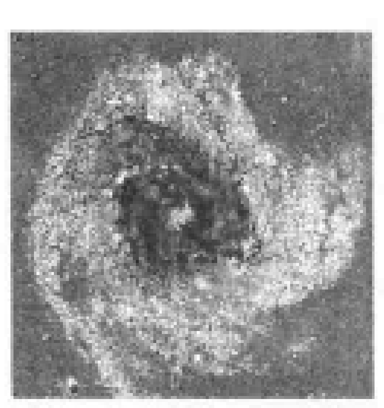
图 5-25 密度分割空间示意图密度分割平面示意图

下面介绍 MATLAB 中的灰度分层法彩色图像的实现，执行结果如图 5-26 所示。

```
%Grayslice 灰度分层法彩色图像处理
clc;
%I=imread('nego4024.tif');
%I=imread('moon.tif');
I=imread('m83.tif');
imshow(I);
title('originalimage')
X=grayslice(I,16);
figure,imshow(X,hot(16));
title('graysliceimage')
```



（a）原始图像



（b）处理后图像

图 5-26　灰度分层法彩色图像处理

2. 空间域灰度级—彩色变换

空间域灰度级—彩色变换是一种更为常用的、比密度分割更有效的伪彩色增强法。它是根据色度学的原理，将原图像的灰度分段经过红、绿、蓝三种不同变换，变成三基色分量，然后用它们分别去控制彩色显示器的红、绿、蓝电子枪，便可以在彩色显示器的屏幕上合成一幅彩色图像。彩色的含量由变换函数的形状而定。典型的变换函数如图 5-27 所示，其中图（a）、（b）、（c）分别为红、绿、蓝三种变换函数，而图（d）是把三种变换画在同一张图上以便看清相互间的关系。由图（d）可见，只有在灰度为零时呈蓝色，灰度为 $L/2$ 时呈绿色，灰度为 L 时呈红色，灰度为其他值时将由三基色混合成不同的色调。

下面沿用第二节中红外目标识别的应用例子，应用空间域灰度级—彩色变换的方法，进行图像增强，实现例程如下：

```
clear all;
a=imread('E:\MATLAB7\work\5-2\yuantu.bmp'); %图像读入
figure,imshow(a,[]); %显示图像
figure,imhist(a,64) %直方图统计显示
c=zeros(size(a));
pos=find(a>=60&a<=105);
c(pos)=a(pos);
```

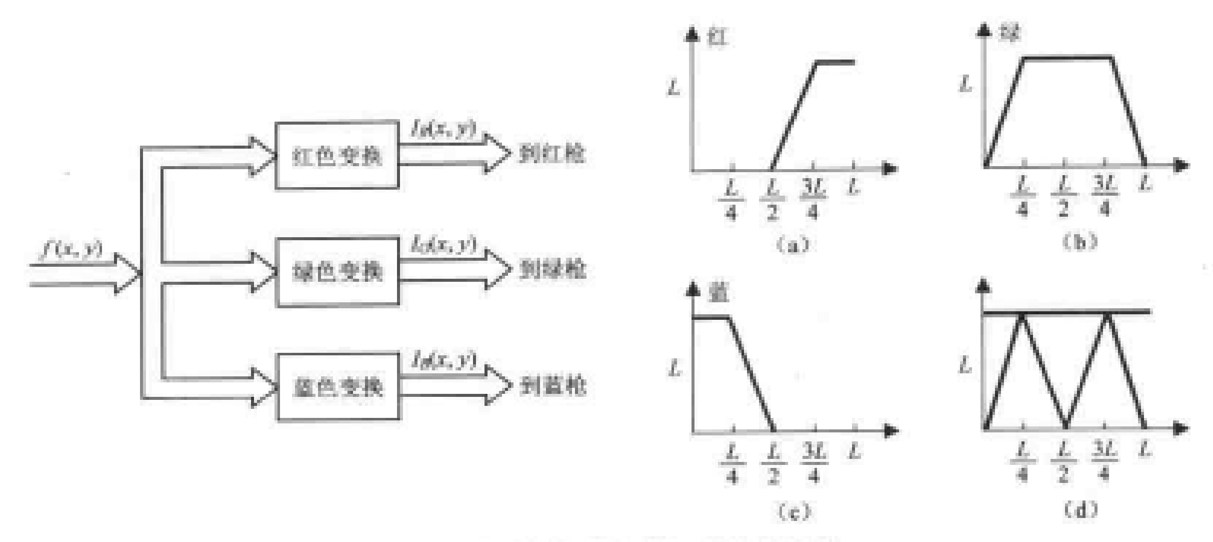
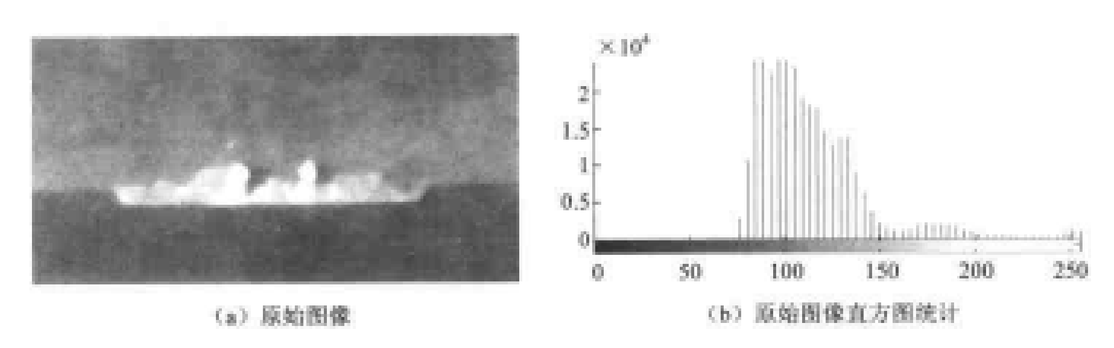


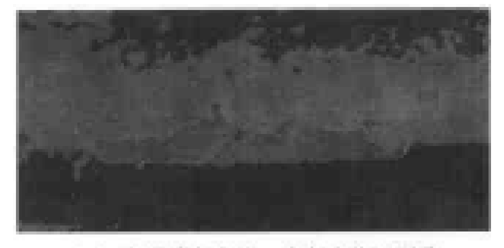
图 5-27　典型的空间域灰度级—彩色变换函数

```
b(:,:,3)=c;
c=zeros(size(a));
pos=find((a>=105)&(a<150));
c(pos)=a(pos);
b(:,:,2)=c;
c=zeros(size(a));
pos=find(a>=150);
c(pos)=a(pos);
b(:,:,1)=c;
%空间域灰度级—彩色变换
b=uint8(b);
figure,imshow(b,[]); %显示变换后图像
```

图 5-28 显示了原始图像、原始图像直方图统计和进行变换后的图像。



（a）原始图像　　　　（b）原始图像直方图统计



（c）空间域灰度级—彩色变换后图像

图 5-28　空间域灰度级—彩色变换

3. 频率域伪彩色增强

频率域伪彩色增强时先把黑白图像经傅立叶变换到频率域，在频率域内三个不同传递特性的滤波器分离成三个独立分量，然后对它们进行逆傅立叶变换，便得到三幅代表不同频率分量的单色图像，接着对这三幅图像作进一步的处理（直方图均衡化），最后将它们作为三基色分量分别加到彩色显示器的红、绿、蓝显示通道，从而实现频率域分段的伪彩色增强。其框图如图 5-29 所示。

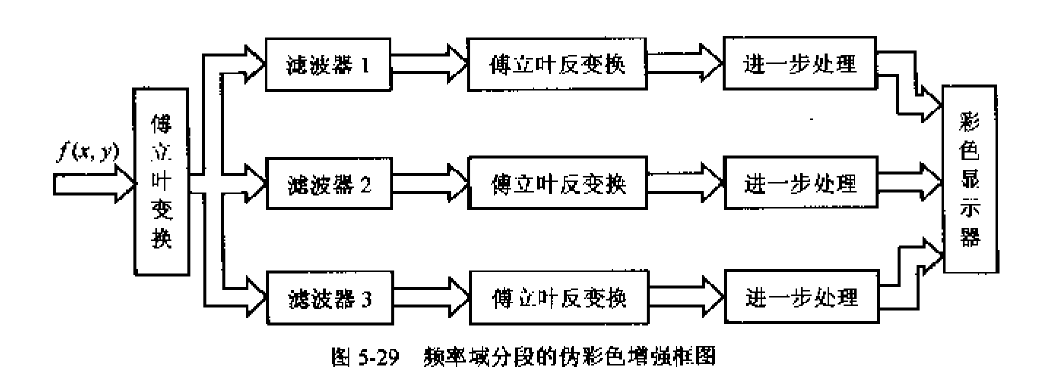


图 5-29 频率域分段的伪彩色增强框图

5.4.3 假彩色增强技术

假彩色增强所处理的对象不是一幅黑白图像，而是一幅自然彩色图像或是同一景物的多光谱图像。通过映射函数变换成新的三基色分量，彩色合成使增强图像中各目标呈现出与原图像中不同的彩色，这种技术成为假彩色增强。假彩色增强目的有两个：一是使感兴趣的目标呈现奇异的彩色或置于奇特的彩色环境中，从而更受人注目；另一个是使景物呈现出与人眼色觉相匹配的颜色，以提高对目标的分辨力。

假彩色增强的一个重要应用是用于多光谱遥感图像。多光谱图像中除了可见光波段图像外，还包括一些非可见光，由于它们的夜视和全天候能力，可得到可见光波段无法获得的信息，因此若将可见光与非可见光波段结合起来，通过假彩色处理，就能获得更丰富的信息，便于对地物识别。

多光谱图像的假彩色增强可表示为：

$$R_F=f_R\{g_1,g_2,\cdots,g_i,\cdots\}$$
$$G_F=f_G\{g_1,g_2,\cdots,g_i,\cdots\}$$
$$B_F=f_B\{g_1,g_2,\cdots,g_i,\cdots\}$$

式中 g_i 表示第 i 波段图像，f_R、f_G、f_B 表示通用的函数运算，R_F、G_F、B_F 为经增强处理后送往彩色显示器的三基色分量，合成一幅假彩色图像。

对于自然景色图像，通用的线性假彩色映射可表示为

$$\begin{bmatrix} R_F \\ G_F \\ B_F \end{bmatrix}=\begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix}\bullet\begin{bmatrix} R_f \\ G_f \\ B_f \end{bmatrix}$$

MATLAB 函数 imfilter 对一幅真彩色（三维数据）图像使用二维滤波器进行滤波，相当

于使用一个两维滤波器对数据的每一个平面单独进行滤波。

使用均值滤波器对真彩图像的每一个颜色平面进行滤波例程：

```
rgb=imread('peppers.png');
h=ones(5,5)/25;
rgb2=imfilter(rgb,h);
%B=IMFILTER(A,H)filtersthemultidimensionalarrayAwiththe
%multidimensionalfilterH.Acanbelogicaloritcanbeanonsparsenumeric
%arrayofanyclassanddimension.Theresult,B,hasthesamesizeandclassas%A.
subplot(1,2,1),imshow(rgb);
title('originalimage')
subplot(1,2,2),imshow(rgb2);
title('color-filteredimage')
```

执行结果如图 5-30 所示。



（a）原始图像



（b）滤波处理后图像

图 5-30　真彩色图像的均值滤波

5.4.4　HSI 变换

HSI 变换也称彩色变换或蒙塞尔（Munsell）变换。在图像处理中通常应用的有两种彩色坐标系（或彩色空间）：一是由红（R）、绿（G）、蓝（B）三原色构成的彩色空间（RGB 坐标系或 RGB 空间）；另一种是由色调（Hue）、饱和度（Saturation）及亮度（Intensity）三个变量构成的彩色空间（HSI 坐标系或 HSI 空间）。也就是说一种颜色既可以用 RGB 空间内的 R、G、B 来描述，也可以用 HSI 空间的 H、S、I 来描述，前者是从物理学角度出发描述颜色，后者则是从人眼的主观感觉出发描述颜色。HSI 变换就是 RGB 空间与 HSI 空间之间的变换。由于 HSI 变换是一种图像显示、增强和信息综合的方法，具有灵活实用的优点，因此产生了多种 HSI 变换式。HSI 变换的作用：

（1）可以进行不同分辨率遥感图像的合成显示；

（2）可以使合成图像更加饱和；

（3）可以通过对亮度的滤波增强图像；

（4）便于多源数据的综合显示。

除此之外，HSI 变换还可以进行其他处理以达到特定的增强和信息提取的目的，如可将色调图像不改，亮度和饱和度置常数，以突出地物色调在空间上的分布。

5.5 图像的代数运算

图像的代数运算是指对两幅或两幅以上输入图像的对应像元逐个进行和、差、积、商四则运算，以产生有增强效果的图像。图像的代数运算是一种比较简单和有效的增强处理，是图像增强处理中的常用方法。这些基本的运算方法在前面已经介绍过，这里重点介绍它们的应用。

5.5.1 加运算

图像相加可以把同一景物地多重影像加起来求平均，以便减少图像的随机噪声，也可以为某种增强效果的需要而把多幅图像有目的叠加在一起。图像相加可以把同一景物的多重影像加起来求平均，以便减少图像的随机噪声，也可以为某种增强效果的需要而把多幅图像有目的地叠加在一起。

5.5.2 减运算

图像的减运算，又称减影技术，是指对同一景物在不同时间拍摄的图像或同一景物在不同波段的图像进行相减。它可以用来消除影像中不希望的附加模式，可以缓慢地改变背景的阴影模式、周期性噪声模式或在影像中每一个像元点上所知的任何附加混杂信息。同时图像相减能用以指导动态监测、运动目标检测和跟踪及目标识别等工作。

5.5.3 乘运算

图像相乘可用来遮掉图像的某些部分。在图像处理中最有用的就是卷积运算，卷积运算一般用作图像轮廓增强。

5.5.4 商运算

图像相除又称比值运算，是遥感图像处理中常用的方法，它是由两个对应像元的灰度值之比或几个波段组合的对应像元灰度值之比获得。利用比值处理可以扩大不同地物的光谱差异，区分在单波段中容易发生混淆的地物，同时可以消除或减弱地形阴影、云影影响和植被干扰以及显示隐伏构造等。

例如有些地物在单波段图像内的亮度差异极小，用常规方法难以区分它们。像水和沙滩，在第四波段和第七波段的亮度非常接近，容易混淆。但如果把两波段图像相除，其比值的差异极大，就很容易将它们区分开。

当然，图像的代数运算除了以上介绍的内容外，还应包括最大值运算、最小值运算、均值运算以及逻辑运算等。运用图像相除应注意它的缺点：

（1）比值使得原来图像的独立波谱意义不存在了，失去了地物的总的反射强度信息，为了补救这一不足，可利用一个波段图像和两个比值图像作彩色合成。

（2）比值处理常常放大了噪声，因而比值处理前应充分做好消除噪声的工作。

【应用实例】

低信噪比条件下运动点目标的检测，必须通过对图像进行累加来提高图像的信噪比，为下一

步进行分割检测提供条件，这是一种典型的图像相加应用实例。下面我们来详细介绍这一应用。

首先我们来模拟产生噪声图像。

定义信噪比 SNR=(TargetAmp-NoiseAmp)/σ，其中 TargetAmp 为目标均值，NoiseAmp 为噪声，NoiseAmp 为噪声均值，σ 为噪声差。

目标点：Noise+SNR*σ

非目标点：Noise

噪声图像由 MATLAB 标准随机函数 randn()生成，噪声和目标线段的产生如下所示：

```
clear all;
Frame=100;%图像序列总长
N=128;%图像尺寸
SNR=2.0;%信噪比
Delta=3;%噪声方差
NoiseAverageAmp=30;%噪声均值
Startx0=30;
Starty0=100;%目标起始点
Vx=0.8;Vy=-0.4%目标速度
TargetAmplitude=SNR*Delta;%信噪比
In_image=zeros(Frame,N,N);
Forfr=1:Frame;
rand(state,sun(100*clock));
In_image(fr,:,:)=NoiseAverageAmp+round(Delta*randn(N)); %产生噪声图像
In_image(fr,Startx0+round((fr-1)*Vx),Starty0+round((fr-1)*Vy))=TargetAmplitude+In_image
(fr,Startx0+round((fr-1)*Vx),Starty0+round((fr-1)*Vy));
%产生目标图像，为噪声图像和目标点相加
```

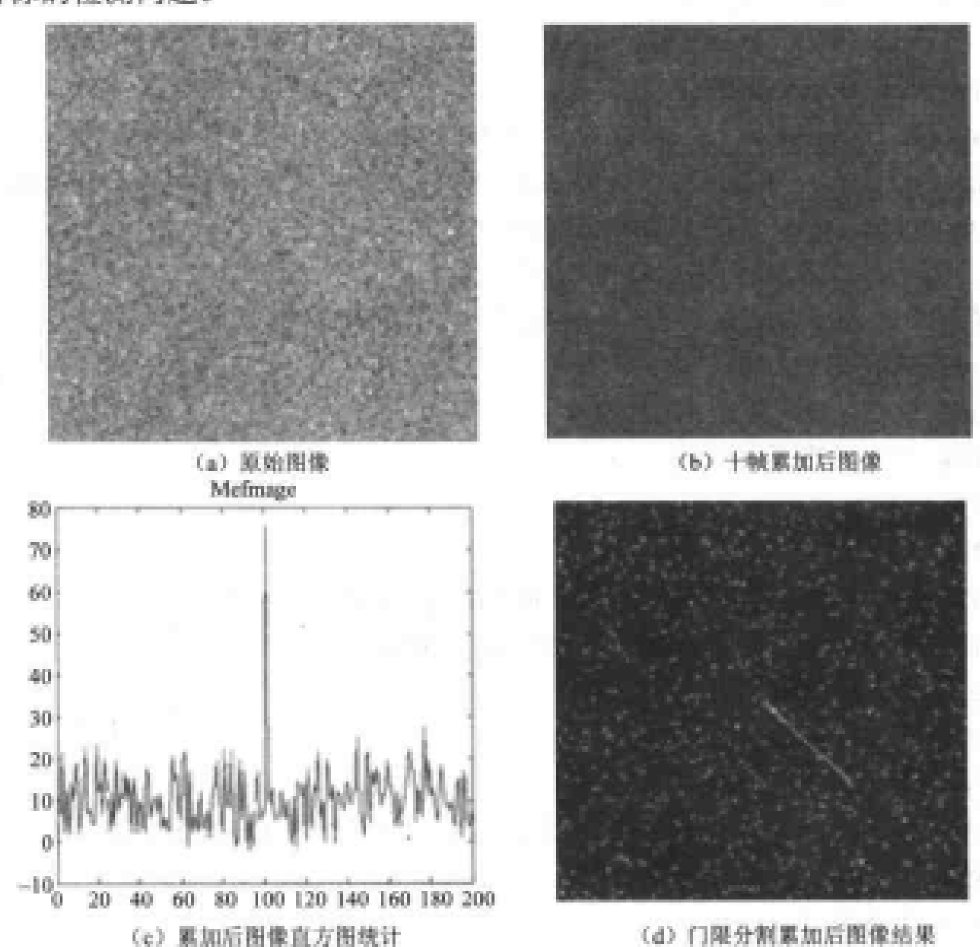
如果采用的是直接累加的算法，实现较简单，实现例程如下：

```
%%%直接帧间累加算法
clear all
N=128;%%%N 为图像的维数
N2=N*N;
ImageNum=10;
MeanNoise=0;%%%MeanNoise 为原始输入图像中的噪声均值
VariantNoise=1;%%%VariantNoise 为原始输入图像中噪声的均方差
SNR=2;%%%SNR 为图像的信噪比
TargetAmplitude=VariantNoise*SNR;
randn('state',sum(100*clock));
NoiseMatrix=VariantNoise*randn(N)+MeanNoise*ones(N);%CreateRandnNoise;
NoiseMatrix(N/2,N/2)=TargetAmplitude+NoiseMatrix(N/2,N/2);
ResultMatrix=NoiseMatrix;
fork=2:ImageNum
h=waitbar(k/ImageNum);
```

```
randn('state',sum(100*clock));%%%resetsthenormalgeneratortoadifferentstateeachtime
%NoiseMatrix=VariantNoise*zeros(N)+MeanNoise*zeros(N);%CreateZerosNoise;
NoiseMatrix=VariantNoise*randn(N)+MeanNoise*ones(N);%CreateRandnNoise;
NoiseMatrix(round(N/2),round(N/2))=TargetAmplitude+NoiseMatrix(round(N/2),round(N/2));
ResultMatrix=NoiseMatrix+ResultMatrix;
end
close(h);
A=NoiseMatrix(N/2,:);
B=ResultMatrix(N/2,:);
figure,mesh(NoiseMatrix);%plot(A);
figure,mesh(ResultMatrix);%plot(B);
```

构造的原始背景图像如图 5-31（a），累加后的图像如图 5-31（b），累加后图像的直方图如图 5-31（c），从直方图中可以看出，通过累加后，图像中目标位置的灰度增强，其他位置高斯白噪声累加后灰度值不能够进行积累，这样就可以提高点目标信噪比，图 5-31（d）为累加后图像进行门限分割所得的图像。

由图 5-31 中各图的比较可以看出，经过能量累加后，信噪比明显提高，也大大增加了目标的检测概率，直接能量累加算法以较小的计算量和存储空间解决了低信噪比下以较快速度运动的点目标的检测问题。



（a）原始图像　（b）十帧累加后图像

（c）累加后图像直方图统计　（d）门限分割累加后图像结果

图 5-31　低信噪比条件下运动点目标的检测

第6章 图像编码

在计算机图像处理系统中，数字图像的数据量是非常大的。例如一幅具有分辨率为640×480的彩色图像，如果每个像素用16位表示，它的数据量约为（640×480×16×3）bit。然而数字图像又具有很大的压缩潜力。因为数字图像中像素与像素之间无论是在行还是列上都有很大的相关性，加之人类的视觉特性对彩色的敏感程度有一定的局限性，对图像中颜色的细微变化是观察不到的，所以完全可以把数字图像中这部分冗余信息去掉，并允许在一定限度失真的前提下，对图像数据进行压缩。

数据压缩是按照某种方法，从给定的信源中推出简化的数据表示，它是通过减少信号空间量的方法使信号能安排到给定的数据样本集中，即去掉冗余度但不会减少信息量。

编码压缩方法有许多种，从不同的角度出发有不同的分类方法，比如从信息论角度出发可分为两大类：

（1）冗余度压缩方法，也称无损压缩，信息保持编码或熵编码。具体讲就是解码图像和压缩编码前的图像严格相同，没有失真，从数学上讲是一种可逆运算。

（2）信息量压缩方法，也称有损压缩，失真度编码或熵压缩编码。也就是讲解码图像和原始图像是有差别的，允许有一定的失真。

还需要说明的是选用编码方法时一定要考虑图像信源本身的统计特征、多媒体系统（硬件和软件产品）的适应能力、应用环境以及技术标准。

数据压缩技术经过几十年的发展，针对不同的原始数据的特点如文本、图像、声音、视频、动画已研究出了不同的编码方法。本章将针对压缩原理，介绍图像数据的几种编码方法。

数据压缩的三个重要指标是：压缩比、压缩算法和失真性。

（1）压缩比：图像压缩前后所需的信息量之比，压缩比越大越好。

（2）压缩算法：即利用不同的编码方法，实现对图像的数据压缩。常用的有LZ编码、游程编码、哈夫曼编码、算术编码、变换编码，等等。

（3）失真性：压缩算法可分为有损压缩和无损压缩。无损压缩可以保证无失真地恢复原始数据，使压缩前后的图像没有差别，但这类压缩的压缩比一般比较小。有损压缩使原始数据不能完全恢复，信息受到一定的损失，但压缩比将会很高，复原后的图像有一定的失真。

6.1 图像的信息量度量和信息冗余

6.1.1 图像的信息量度量

数字图像形成的关键步骤是在空间(x, y)和光亮度f上都进行离散化，通常把这一过程叫做采样与量化。采样与量化处理是决定最终的数字图像与原始图像接近的两个关键因素，也关系到数据量的大与小。

1. 采样

如果对图像进行等间距采样，即在 x 和 y 方向上取 N 个点，并被排成 $N\times N$ 的矩阵，矩阵中的每一个点为离散化的亮度值 $f(x,y)$，它对应于数字图像中的一个元素，称为像素。采样点的多少直接影响到数字图像与原图像的失真程度，而如何表示每一个采样点的亮度值也是导致最终数字图像质量好坏的关键因素。

2. 量化

量化过程就是用有限的离散量代替无限的连续的模拟量的一对多的映射过程。图像的亮度 f 是连续变化的数值，$f(x,y)$的光亮度 L 表示为：$L_{\min}\leqslant L\leqslant L_{\max}$。

$[L_{\min},L_{\max}]$称为灰度级范围。把$[L_{\min},L_{\max}]$分成 K 个等间距的区间，每个区间对应一个亮度值 f_i，这样就有 K 个亮度值，称之为灰度级 K。为了计算机处理的方便，灰度级 K 用 2 的整数幂表示，即 $K=2^m, m=1,2,\cdots,8$。当 m=6 时，$K=2^6=64$ 个灰度级。当$f(x,y)$的光亮度落在第 i 个区域中时，就被合入到区域的中心，量化结果就是这个区域的灰度值f_i。

采样点 N 与量化灰度级 K 的选择确定了数字图像的质量，也决定了数字图像所占有计算机存储空间的大小。例如，一幅单色调灰度级为 64 的 128×128 的图像，需要 128×128×6=98304 位表示。因为 $\log_2 64=6$，每一个像素用 6 位表示。6 称为位深度，位深度越深，表示的灰度就越多。一般情况下，用位深度 8（256 个灰度级）表示单色调图像。如果是一幅 128×128 的彩色图像，每个 RGB 颜色分量都用 256 个灰度级表示，需要 128×128×8×3 位表示，这时每一个像素用 24 位表示（各颜色分量分别用 8 位表示）。

3. 均匀采样与非均匀采样

上面提到的在(x,y)方向上等间距的采样成为均匀采样，而非均匀采样则是在图像细节少的区域采用较稀疏的采样，在细节比较多的区域采用密集采样，这样获得的图像信息量没有减少，但是数据量却有效地降低了。需要指出的是，分配采样点的时候，应该在灰度变化的边界标上非均匀采样标志。

4. 线性量化和非线性量化

将表示数字图像的灰度级范围分为等间隔的子区间叫做线性量化，而非线性量化是指将灰度级范围分为不等间隔的子区间。与均匀采样和非均匀采样的概念一致，对灰度级出现频率比较高的区间，量化区间变窄，而另一些灰度级出现频率较低的范围量化区间变宽。

5. 图形的信息量度量

图像的信息量主要取决于图像的分辨率和像素的位深度。图像的分辨率由(x,y)方向的采样点数决定。如果(x,y)方向等间隔采样 N 个点，则图像的分辨率为 $N\times N$，像素的位深度就是由量化级确定的。如果用 256 个灰度级表示单色调图像，则每个像素的位深度为 8 位。

6.1.2 数字图像的信息冗余

冗余是指信息中存在着多余的数据。数字化后的图像就存在着大量的信息冗余，分为空

间冗余、信息熵冗余、结构冗余和视觉冗余等。下面简单介绍视觉冗余和空间冗余。

1. 视觉冗余

人的眼睛对图像细节和颜色的辨认有一个极限，人的视觉特性决定了他最多只能分辨出2的16次幂种颜色，而彩色图像一般每个像素用24位表示，则可表示出2的24次幂种颜色，由此而来的数据冗余称为视觉冗余。

2. 空间冗余

图像中的大部分景物，其表面颜色都是均匀的、连续的。把图像数字化成像素点的矩阵后，大量相邻像素的数据是完全一样或十分接近的，这就是空间冗余。

6.1.3 图像的有损编码和无损编码

图像编码是指按照一定的格式存储图像数据的过程，而编码技术则是研究如何在满足一定的图像保真条件下，压缩表示原始图像数据的编码方法。目前有很多流行的图像格式标准，如BMP、PCX、TIFF、GIF、JPEG等等，采用不同的编码方法，一般可以将其分为有损编码和无损编码两类。

1. 无损编码

无损编码指对图像数据进行了无损压缩，解码后重新构造的图像与原始图像之间完全相同。行程编码就是无损编码的一个实例，其编码原理是在给定的数据中寻找连续重复的数值，然后用两个数值（重复数值的个数，重复数值本身）取代这些连续的数值，以达到数据压缩的目的。运用此方法处理拥有大面积色调一致的图像时，可达到很好的数据压缩效果。常见的无损压缩编码有：

- 哈夫曼编码；
- 算术编码；
- 行程编码；
- Lempel zev 编码。

2. 有损编码

有损编码是指对图像进行有损压缩，致使解码后重新构造的图像与原图像之间存在着一定的误差。有损压缩利用了图像信息本身包含的许多冗余信息，例如视觉和空间冗余。

针对人类的视觉对颜色不敏感的生理特性，对丢失一些颜色信息所引起的细微误差不易被发现的特点来删除视觉冗余。又由于图像信息之间存在着很大的相关性，存储图像数据时，并不是以像素为基本单位，而是存储图像中的一些数据块，以删除空间冗余。

由于有损压缩一般情况下可获得较好的压缩比，因此在对图像的质量要求不苛刻的情况下是一种理想的编码选择，常见的有损编码有：

- 预测编码，如DPCM，运动补偿编码；
- 频率域方法，如正文变换编码(如DCT)，子带编码；
- 空间域方法，如统计分块编码；

- 模型方法编码，如分形编码，模型基编码；
- 基于重要性编码，如滤波，子采样，比特分配，矢量量化。

6.1.4 哈夫曼编码技术

哈夫曼编码是运用信息熵原理的一种无损编码。压缩方法是利用变长编码将图像中出现概率比较大的灰度值赋予短码字，而对出现概率小的灰度值赋予长码字，从而达到压缩数据的目的。

例如，一幅40个像素的图像，具有5个灰度级A、B、C、D、E，如果40个像素中，A级具有15个像素，B级具有7个像素，C级具有7个像素，D级具有6个像素，E级具有5个像素。对每个像素编码就需要3位（$\log_2 5 \approx 3$），40个像素需要120位。如果求出图像的信息熵：

$$H(x)=\sum_{i=1}^{n} p(x)\log_2(\frac{1}{p(x)})=(15/40)\log_2(40/15)+(7/40)\log_2(40/7)+...$$

$$+(5/40)\log_2(40/5)=2.1895$$

则每个像素平均所占位数是2.1898位，40个像素所用的总位数是40×2.1895=87.84≈90，这样就产生了1.3/1的压缩比。

哈夫曼编码方法可以归纳为以下几步：

- 将图像灰度按照概率大小排列；
- 把两个最小的概率加起来作为新的概率；
- 重复步骤（1）、（2），直到概率之和达到1为止；
- 每次合并符号时，将被合并的符号赋以1和0（大概率赋1，小概率赋0，或相反）；
- 寻找从每一信源符号到概率为1处的路径，记录下路径上的1和0；
- 对每一符号写出“1”、“0”序列，序列的顺序是从树根到信源符号节点。

如果按此步骤对上面所给出的例子进行哈夫曼编码，结果如图6-1所示。

各个灰度级的出现概率从大到小依次为：A-0.375,B-0.175,C-0.175,D-0.150,E-0.125 。所形成的哈夫曼编码表如表6-1所示。

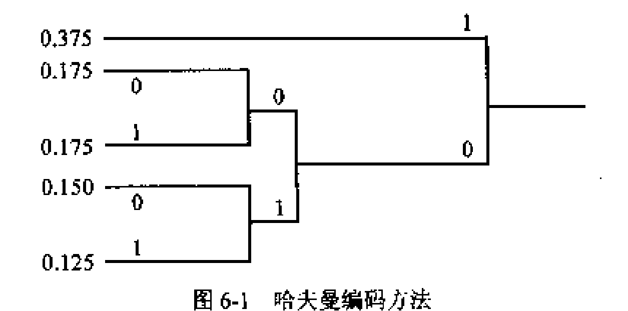


图6-1　哈夫曼编码方法

表6-1　　哈夫曼编码表

码　字	灰　度　级	码　长
1	表示A色	1
000	表示B色	3
001	表示C色	3
010	表示D色	3
011	表示E色	3

哈夫曼编码的特点：

利用哈夫曼编码压缩图像数据时，必须读取图像数据两次。第一次读取数据计算每个数

据出现的频率，并对各数据出现的频率以二叉树方式进行排序，在排序过程中获取各数值的编码值。将这些长度不等的编码值与对应的图像数据放置到一个转换表格中。第二次读取数据是利用转换表格中的编码值取代图像数据存入图像文件中。

此外哈夫曼编码还具有以下特点：

（1）哈夫曼编码构造的数据不一定是唯一的。原因是在给两个最小概率的图像灰度值进行编码时，可以是大概率为 1，小概率为 0，也可以相反。而当两个灰度值的概率相等时，“1”、“0”的分配也是随机的，这就造成了编码的不唯一性。

（2）当图像灰度值分布不是很均匀时，哈夫曼编码的效率就高。当信源概率是 2 的负幂次方时，编码效率为 100%。而在图像灰度值的概率分布比较均匀时，哈夫曼编码的效率就很低。

（3）哈夫曼编码需要先计算出图像数据的概率特性形成编码表后，才能对图像数据编码，因此哈夫曼编码缺乏构造性，即不能使用某种数学模型建立信源符号与编码之间的对应关系，而必须通过查表方法，建立起他们之间的对应关系。如果信源符号很多，编码表就会很大，这必将影响到存储、编码和传输。通常可以在经验基础上预先提供哈夫曼编码表，但性能将有所下降。

下面用 MATLAB 实现哈夫曼编码。程序将输入的数据（数字矩阵）进行哈夫曼编码，然后反编码，判断是否是无失真编码，最后给出压缩前后存储空间的比较。

```
clear all;
fprintf('Reading data….')
data=imread('E:\temp\originalimage(holl).bmp')
data=uint8(data);% 读入数据，并将数据限制为 unit8 型
fprintf('Done!\n')
fprintf('Comqressing data…')
[zipped,info]=norm2huff(data);  %进行压缩编码
fprintf('Done!\n')
fprintf('Decomqressing data…')
unzipped=huff2norm(zipped,info);% 进行解压缩
fprintf('Done!\n')
isOK=isequal(data(:),unzipped(:)) %显示压缩效果
whos data zipped unzipped
%%%%%%%%%%    norm2huff    %%%%%%%%%%
% NORM2HUFF 哈夫曼编码器
% 对于输入向量，NORM2HUFF(X) 返回向量的哈夫曼编码后的码串
%  矩阵用 X(:)形式输入
%  输入限制为 unit8 格式，输出为 unit8 的序列
%  […,info] 返回解码需要的解码信息
% INFO.pad =最后添加的比特数
% INFO.huffcodes=Huffman 码字
% INFO.ratio=压缩率
% INFO.length=原始数据长度
```

```
% INFO.maxcodelen=最大码长
function   [zipped, info]   = norm2huff(vector)
if ~isa(vector,'unit8'),
    error ('input argument must be a unit8 vector');
end                          %保证输入为 unit8 的数据
vector = vector(:)';         %将输入向量转换为行向量
f=frequency(vector);         %计算个元素出现的概率
simbols=find(f~=0);          %寻找数出现的所有元素
f=f(simbols);
[f,sortindex]=sort(f);       %将元素按照出现概率排列
simbols=simbols(sortindex);
len=length(simbols);
simbols_index=num2cell(1:len);
codeword_tmp=cell(len,1);
while length(f)>1,
    index1=simbols_index{1};
    index2=simbols_index{2};
    codeword_tmp(index1)=addnode(codeword_tmp(index1),unit8(0));
    codeword_tmp(index2)=addnode(codeword_tmp(index1),unit8(1));
    f=[sum(f(1:2))    f(3:end)];
    simbol_index=[{[index1 index2]} simbols_index(3:end)];
    %将数据重新排列，是两个节点的频率尽量与前一个节点的频率相当
    [f,sortindex]=sort(f);
    simbols_index=simbols(sortindex);
end  %对应相应的元素和码字
codeword=cell(256,1);
code(simbols)=codeword_tmp;  %计算总的字符串长度
len=0;
for index=1:length(vector),
    len=len+length(codeword{double(vector(index))+1});
end     %产生 0 1 序列
string=repmat(unit8(0),1,len);
pointer=1;
for index=1:length(vector),
    code = codeword{double(vector(index))+1};
    len=length(code);
    string(pointer+(0:len-1))=code;
    poiter=pointer+len;
end
%如果需要的话加零
```

```
len=length(string);
pad=8-mod(len,8);
if pad>0,
      string =[string unit8(zeros(1,pad))];
end
%保存实际有用的码字
codewod=codeword(simbols);
codelen=zeros(size(codeword));
weights=2.^(0:23);
maxcodelen=0;
for index=1:length(codeword),
      len=length(codeword{index});
      if len>maxcodelen,
            maxcodelen=len;
      end
      if len>0,
            code=sum(weights(codeword{index}==1));
            code=bitset(code,len+1);
            codeword(index)=code;
            codelen(index)=len;
      end
end
codeword=[codeword{:}];
%计算压缩后的向量
cols=length(string)/8;
string=reshape(string,8,cols);
weights=2.^(0:7);
zipped=uni8(weights*double(string));
%存储数据到一个稀疏矩阵
huffcodes=sparse(1,1);
for index=1:numel(codeword),
      huffcodes(codeword(index),1)=simbols(index);
end
%产生信息结构体
info.pad=pad;
info.huffcodes=huffcodes;
info.rato=cols./length(vector);
info.length=length(vector);
info.maxcodelen=maxcodelen;
%%%%%%%%%%    addnode %%%%%%%%%%%%%%
```

```
function codeword_new=addnode(codeword_old,item)
codewod_new=cell(size(codeword_old)),
codeword_new{index}=[item codeword_old{index}];
%%%%%%%%      huff2norm       %%%%%%%%%%%%
function vector =huff2norm(zipped,info)
% HUFF2NORM    哈夫曼解码器
if ~isa(zipped,'unit8'),
    error('input argument must be a unit vector')
end    %产生 01 序列
len=length(zipped );
string=repmat(unit8(0),1,len.*8);
bitindex=1:8;
for index =1:len,
    string(bitindex+8.*(index-1))   ,
    unit8(bitget(zipped(index),bitindex));
end     %调整字符串
string=logical(string(:)');
len=length(string);
string((len-info.pad.1+1):end)=[ ];
len=length(string);    %解码
weights=2>^(0:51);
vector=repmat(unit8(0),1,info.length);
vectorindex=1;
codeindex=1;
code=0;
for index=1:len,
    code=bitset(code,codeindex,string(index));
    codeindex=codeindex+1;
    byte=decode(bitset(code,codeindex),info);
    if byte>0,
        vector(vectorindex)=byte-1;
        codeindex=1;
        code=0;
        vectorindex=vectorindex+1;
    end
end
%%%%%%%%%%%%%%      decode        %%%%%%%%%%
function byte=decode(code,info)
byte=info.huffcode(code);
%%%%%%%%%%     frequence   %%%%%%%%%%
```

```
function f=frequency(vector)
% FREQUENCY 计算元素出现概率
if ~isa(vector,'unit8'),
    error('input argument must be a unit8 vector ')
end
f=repmat(0,1,256);  %扫描向量
len=length(vector);
for index=1:255,
   f(index+1)=sum(vector==unit8(index));
end  %归一化
f=f./len;
```

运行上述程序，得到结果为：

Name	Size	Bytes	Class
data	256*25665535	unit8 arrray	
unzipped	1*65535	65535	unit8 arrray
zipped	1*57712	57712	unit8 arrray

Grand total is 188784 elements using 188784 bytes

其中压缩的信息结构体 info 为：

```
        pad:7
  huffcodes:[108471*1  double]
      ratio:0.8806
     length:65535
maxcodelen:16
```

6.1.5 行程编码技术

行程编码也叫做 RLE 压缩方法，其中 RLE 是 run-length-encoding 的缩写，这种压缩方法广泛应用于各种图像格式的数据压缩处理中，是最简单的压缩图像方法之一。

行程编码的原理是在给定的数据图像中寻找连续的重复数值，然后用两个字符取代这些连续值。例如，一串字母表示的数据为“aaabbbbccccdddeeddaa”，经过行程编码处理可表示为“3a4b4c3d2e2d2a”。这种方法在处理包含大量重复信息的数据时可以获得很好的压缩效率。但是如果连续重复的数据很少，则难获得较好的压缩比，甚至压缩后的编码字数大于处理前的图像字节数。因此行程编码的压缩效率与图像数据的分布情况密切相关。

另外，由于利用应用程序实现行程编码的压缩时，分别用一个字节存放重复值，用另一个字节存放重复值连续的次数，然后利用这两个字节代替这串连续重复出现的同一个数据。因此，通常需要在表示重复次数的字节中利用前一位或二位作为标志位，提示应用程序注意随后的几位表示的是连续重复的次数，另一字节的编码则表示重复数据的值。两个字节的编码最多只能取代 63 个(2^6-1)或 127 个(2^7-1)字节长度的重复数据值。如果超出了所能表示的长度，则通过增加字节数的方式给予解决。

6.2 典型的图像限失真压缩编码方法

6.2.1 图像预测编码技术

在图像编码之前对编码图像进行预测，并将预测差输出，供量化编码，而在接收端将预测差的码字解码后再与预测值相加以恢复原图，这种方法就称为预测编码。若收发两端的预测器完全相同，而且发端不经过量化，那么在接收端完全可以无失真地恢复图像。预测映射变换本身并不引入误差。然而在实际应用中常常使用量化器以提高压缩比，因此，在接收端经解码器得到的预测差存在量化误差，最终恢复的图像会有失真。

预测编码有线性和非线性两类。它们可以在一幅图像内进行（帧内预测编码），也可以在多幅图像之间进行（帧间预测编码）。预测编码没有图像数据的空间和时间冗余特性，用相邻的已知像素来预测当前像素的取值，然后再对预测误差进行量化和编码。预测编码的关键在于预测算法的选取，这与图像信号的概率分布很有关系，实际中常根据大量的统计结果采用简化的概率分布形式来设计最佳的预测器。预测编码法中最重要的是线性预测法，通常也称为“差值脉冲编码调制法”（DPCM）。由于预测公式是线性的，且线性预测法较为简单，易于硬件实现，因而在图像压缩编码中得到广泛应用。现有的图像编码国际标准都采用了DPCM，有人称DPCM“古老简单却有效”，“充满青春活力”。缺点是对信道噪声及误差很敏感，会产生误码扩散，使图像质量大大下降。同时帧内DPCM的编码压缩比很低，因此很少独立使用。帧间预测编码主要利用活动图像序列帧间的相关性，即图像数据的时间冗余来达到压缩的目的，可以获得比较高的压缩比，是消除帧间相关性的主要手段。

图6-2是DPCM的原理框图，为便于分析，把像素按某种次序排成一维序列（如按电视扫描次序）表示某个像素的灰度值。

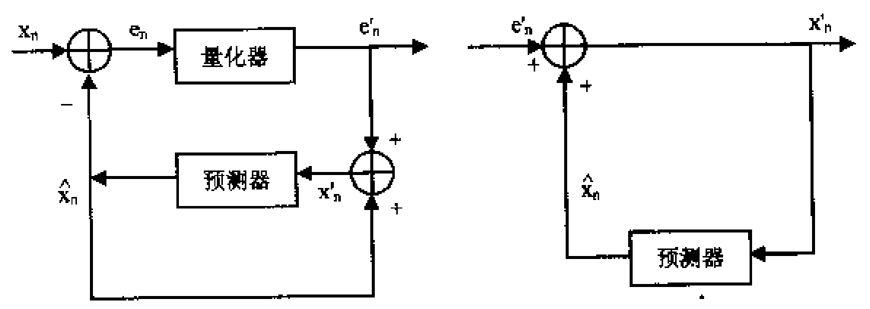


图6-2 DPCM编码（左）解码（右）原理图

以上框图工作过程如下：

（1）x_n与发端预测器产生的预测值$\hat{x}_n$相减得预测差E_n。

（2）e_n经量化后变为$e_n^{'}$，同时引入量化误差。

（3）$e_n^{'}$再经过编码器编成码字(如Huffman码)发送，同时又将$e_n^{'}$加上$\hat{x}_n$恢复输入信号。因存在量化误差，$x_n \neq \hat{x}_n$，但相当接近。发端的预测器及其环路作为发端本地解码器。

（4）发端预测器带有存储器，将$x_{n-1}^{'},\cdots,x_{n-m}^{'}$存储起来以供对$x_n$进行预测得到$\hat{x}_n$。

（5）继续输入下一个像素，即 $n=n+1$，重复上述过程。

接收端和发送端的本地解码部分动作完全一样。若不存在传输误码，则接收端的环路工作和发送端的“小环”完全相同。

预测器的设计是 DPCM 的关键，预测愈准，σ_m^2 愈小，压缩倍数愈高。预测器可以是固定的，也可以是自适应的；可以是线性的，也可以是非线性的。

由于 DPCM 带有量化环节，是个带反馈的非线性系统，对它进行严格分析是相当困难的。实际中常用简化的分析方法对预测器和量化器进行分析，得到局部最优解。

在线性预测中，预测值 $\hat{x}_n$ 是 $x'_{n-1},\cdots,x'_{n-m}$ 的线性组合，即

$$\hat{x}(n)=\sum_{k=1}^{m}a_k x(n-k)$$

式中，a_k 为预测系数，m 为预测阶数。分析可知，须选择适当的 a_k 使得预测误差最小。这是一个求取最佳线性预测的问题。

不失一般性，设 $x(n)$是期望 $E(x(n))=0$ 的广义平稳随机过程，则

$$\sigma_m^2=E\{e_n^2\}=E\{[x(n)-\sum_{k=1}^{m}ax(n-k)]\}$$

为了使 σ_m^2 最小，必定有 $\dfrac{\sigma_m^2}{\partial a_i}=0$

设 $x(n)$的自相关函数为：$R(k)=E\{x(n)x(n-k)\}$ 且 $R(-k)=R(k)$

将其带入上式，可以得

$$\begin{bmatrix} R(0) & R(1) & \cdots & R(m-1) \\ R(1) & R(0) & \cdots & R(m-2) \\ \vdots & \vdots & \vdots & \vdots \\ R(m-1) & R(m-2) & \cdots & R(0) \end{bmatrix}\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{bmatrix}=\begin{bmatrix} R(1) \\ R(2) \\ \vdots \\ R(m) \end{bmatrix}$$

上式左边的矩阵是 $x(n)$的相关矩阵，为 Toeplitz 矩阵，因此用 Levinson 算法可解出各 a_k，从而得到在均方差最小意义下的最优线性预测，以最小熵为准则进行预测也是常用的方法。

线性预测可以减小方差

$$\sigma_m^2=R(0)-\sum_{k=1}^{m}a_k R(k)$$

这里不加证明地给出如上所示的预测方差表达式。由于 $E(x(n))=0$，R(0)即 $x(n)$的方差，可见 $\sigma_m^2<\sigma_{xn}^2$，所以传递差值比直接传递原始信号更有利于数据压缩。$R(k)$愈大，即 $x(n)$的相关性越大，则 σ_m^2 越小，所能达到的压缩比就愈大，当 $R(k)=0\,(k>0)$时，即相邻点不相关时，$\sigma_m^2=\sigma_{xn}^2$，此时预测并不能提高压缩比。

二维线性预测的情况与一维完全类似。设原始图像用 $f(m,n)$来表示，则二维线性预测公式为

$$f(m,n)=\sum_{(k,j\in z)}\sum a_{kj}f(m-k,n-l)$$

其中，a_{kl} 为二维预测系数，z 定义了预测区域，一般取为(m,n)点的邻域，但不包括(m,n)点本身。与一维情况完全类似，上式中系数 a_{kl} 的求取由相关矩阵运算获得，即

$$R(i,j)-\sum_{(k,j)\in z}\sum a_{kj}R(k-i,l-j)=0$$

预测差的方差为：

$$\begin{aligned}\sigma_{emn}^2&=E\{[f(m,n)-\sum_{(k,j)\in z}\sum a_{kj}R(k-i,l-j)=0]^2\}\\&=E\{f(m,n)[f(m,n)-\sum_{(k,j)\in z}\sum a_{kj}R(k-i,l-j)=0]\}\\&=R(0,0)-\sum_{(k,j)\in z}\sum a_{kj}R(k,l)\end{aligned}$$

这与一维时一样，若 $R(k,l)$大，即原图各像素间相关性大，则预测差的方差较小，压缩比可达到很高。同样，若预测域达到某个范围以后，各预测差已不相关，即

$$E\{e(m,n)e(m+i,m+j)\}=0;i,j\neq 0$$

那么，再加大预测区域也不会使预测差的方差下降。

6.2.2 图像变换编码技术

1. 变换编码的基本原理

限失真图像编码中另一类有效的方法是变换编码。在变换编码的通用模型（如图 6-3 所示）中，若用某种形式的正交变换来实现此框图中的映射变换，则这种编码方式称为变换编码。

$f(t)$ → 映射变换 → 量化器 → 编码器 → $s(n)$

图 6-3 变换编码的通用模型

图像信号一般具有较强的相关性，如果所选用的正交矢量空间的基矢量与图像本身的主要特征很接近，那么在这种正交变换矢量空间中描述这一图像信号将会更简单些。从本质上讲，图像经过正交变换后之所以能够实现数据压缩，是因为经过多维坐标系适当的旋转变换后，把散布在各个原坐标中的原始图像数据集中到新的坐标轴上了，从而为后继的量化和编码提供了高效压缩数据的可能性。

为了保证平稳性和相关性，同时也为了减少运算量，在变换编码中，一般在发送端的编码器中，先将原图 $f(m,n)$分成子像素块，然后对每个子像素块进行正交变换，形成变换域中的系数阵列 $F(s,t)$样本。（系数）选择器再选择其中的若干个主要分量进行量化、编码和传输。接收端解码器经解码、反量化后得到带有一定量化失真得变换系数 $f(s,t)$，再经过反变换就得到复原图像 $f(m,n)$。显然，复原图像也带有一定量化失真，但只要系数选择器和量化编码器设计得好，这种失真可限制在允许的范围内。因此变换编码是一种限失真编码。

DCT 相比其他形式的正交变换而言，在其边界处无跳变，能量更集中。它有固定基以及成熟的快速算法，因而成为变换法的主流。现有的国际编码标准均采用了 DCT。

2. 离散余弦变换

DCT 编码属于正交变换编码方式，用于去除图像数据的空间冗余。变换编码就是将图像光强矩阵（时域信号）变换到系数空间（频域信号）上进行处理的方法。在空间上具有强相

关的信号，反映在频域上是在某些特定的区域内能量常常被集中在一起，或者是系数矩阵的分布具有某些规律。我们可以利用这些规律在频域上减少量化比特数，达到压缩的目的。该变换的 MATLAB 工具函数实现在图像变换中已经介绍过了，为说明方便，这里只简单介绍一下离散余弦变换的性质。

DCT 有一个重要性质，就是它的变换矩阵的基向量很近似于 Toeplitz 矩阵（即沿对角线方向的元素都相同）的特征向量。这就是说，对 Toeplitz 矩阵 BGT 将非常接近 K-L 变换。统计表明，人类的语言、图像等信号的自相关矩阵常常表现出具有 Toeplitz 矩阵的特点。从这个意义上来说，DCT 是比较适合对语言、图像等信号作变换处理的一种变换，它的性能接近于 K-L 变换。

DCT 还具有以下性质：

（1）二维正向或反向离散余弦变换是可分离的，因此正向或反向变换能够逐次应用一维变换计算。

（2）离散余弦变换有快速算法。从一维离散余弦变换关系式可以看出，可以利用快速算法。如只有偶数部分，计算更为方便。

图像经 DCT 变换以后，DCT 系数之间的相关性已经很小，而且大部分能量集中在少数系数上，因此，DCT 变换在图像压缩中非常有用，是有损图像压缩国际标准 JPEG 的核心。从原理上讲可以对整幅图像进行 DCT 变换，但由于图像各部位上细节的丰富程度不同，这种整体处理的方式效果不好。为此，发送者首先将输入图像分解为 8×8 或 16×16 的块，然后再对每个图像块进行二维 DCT 变换，接着再对 DCT 系数进行量化、编码和传输；接收者通过对量化的 DCT 系数进行解码，并对每个图像块进行二维 DCT 反变换，最后将操作完成后所有的块拼接起来构成一幅单一的图像。对于一般的图像而言，大多数 DCT 系数值都接近于 0，可以去掉这些系数而不会对重建图像的质量产生重大影响。因此，利用 DCT 进行图像压缩确实可以节约大量的存储空间。

3. Karhunen-Loeve 变换

K-L 变换是 Karhunen-Loeve 的简称，由于该变换去相关最彻底，且由此引出的数据压缩编码产生的均方误差最小，因此被称作最佳变换。

（1）K-L 变换的定义

对于一个离散信号序列，若每一个信号由 n 个样点组成，那么就可以将该信号视作一个 n 维空间的点，而每个样值代表 n 维信号矢量信号的一个分量，记作 $X=(x_1,x_2,\cdots,x_n)^T$ 。向量 X 的协方差矩阵可表示为

$$\phi_x=\begin{bmatrix}\phi_{11} & \phi_{12} & \cdots & \phi_{1n}\\ \phi_{21} & \phi_{22} & \cdots & \phi_{2n}\\ \vdots & \vdots & \vdots & \vdots\\ \phi_{m1} & \phi_{m2} & \cdots & \phi_{mn}\end{bmatrix}$$

其中定义 $\phi_{ij}=E[(x_i-Ex_i)(x_j-Ex_j)]$ 。显然，ϕ_{ij} 是一个对称矩阵。它反映了信号各分量之间的相关性，若各分量之间互不相关，则只有对角线元素不为零，且他们代表了各分量方差。

我们用这个正交矩阵 Q 对信号作正交变换得

$$Y=QX$$

这个变换的基本思想是由 Karhunen 与 Loeve 分别提出的，故被称为 K-L 变换，Q 是 K-L 变换矩阵。

（2）K-L 变换的性质

- 去相关性，即 K-L 变换使变换后的矢量信号 Y 的各分量之间互不相关。
- 能量集中特性，指对 n 维矢量信号进行 K-L 变换后，最大的方差将集中在前 m 个分量之中的这样一种特性($m<n$)。
- 最佳性，指 K-L 变换是均方误差测量下，失真最小的一种变换，故又称最佳变换。其失真量为被略去的各分量的方差之和。
- 尚没有快速算法。关键是对协方差矩阵的特征值和特征向量的求解有困难。解决的方法，一是继续寻找有效的算法;另一方面则寻找性能稍逊但却容易实现的算法。上面介绍的 DCT 就是其中之一。

6.3　数据压缩国际标准

20 世纪 80 年代初期，电视会议系统开始出现，由于码率高，传输线路成本居高不下，阻碍了推广应用。1984 年根据有关建议，国际上成立了“电视电话专家组”，研究在通信领域传输可视电话、会议电视的压缩方案，截至 1990 年底，原 CCITT 通过了多项有关视频压缩编码基础技术的建议，其中具有代表性的是 H.261 建议。H.261 采用了 $P\times64.5$kbit/s 的标称码率，码率从 40kbit/s 到 2Mbit/s（P=30），信源编码采用了帧内/帧间自适应预测和离散余弦变换（DCT），为在现有的不同彩色电视制式之间进行图像通信，又建立了世界通用的 CIF（中间格式）和 QCIF（四分之一普通立即格式）。

与此同时，国际标准化组织（ISO）在 1986 年底成立了“联合图像专家组”（JPEG），对主要用于计算机的静止图像压缩编码进行研究。1991 年 3 月完成了 ISOCD10918 号标准，即 JPEG 标准。它主要使用了两种编码方式：离散余弦变换和差值脉冲编码方式。JPEG 是一种压缩效率较高、适用性强而计算复杂度较低的图像压缩方法。由于其采用帧内编码方式，因此在一段时间内，电视节目制作中的非线性编辑系统较多地采用它进行图像的压缩编码。JPEG 由于其算法的局限性，不适应于快速运动图像的实时压缩编码。

MPEG 即“运动图像专家组”，成立于 1988 年 10 月，隶属于国际标准化组织。这个专家组建立了一系列运动图像和音频压缩编码标准，广泛应用于数字存储、图像通信、广播电视等领域。MPEG 专家组成立的最初目标是把运动图像和音频信号压缩为 1.5Mbit/s 的数据流，供介质存储和消费类应用，该标准主要内容于 1993 年通过，称为 MPEG-1。MPEG-1 水平和垂直分辨率为现行电视的一半，图像质量和 VHS（特高速度）相当，由于采用较低的分辨率和采用运动补偿、DCT、可变字长等编码方式，因此 MPEG-1 达到了较高的压缩比。我国最早采用 MPEG-1 标准，制造了 VCD 影碟机，并形成了巨大市场。

在 MPEG-1 制定过程中，20 世纪 90 年代美国提出高清晰度电视（HDTV）信号数字化并压缩编码用于地面无线广播，这一大胆设想震动了全球广播电视科技工作者，并迅速推动了电视数字化和图像压缩编码研究的发展。在此背景下，MPEG 专家组在 1992 年开始制定广播级 CCIR 601 建议的码率压缩标准和 HDTV 的码率压缩标准，分别为 MPEG-2、MPEG-3，最后高清晰度电视的 MPEG-3 标准并入了 MPEG-2 中的高 1440 级中。MPEG-2 标准的主要

部分在 1996 年通过。MPEG-2 采用了更多的和更细的压缩编码技术，主要有 DCT、运动预测、运动补偿和霍夫曼编码。DCT 技术大大减少了图像中的空间冗余；运动预测和运动补偿大大减少了时间方面的冗余；而霍夫曼编码在信息表示方面提高了编码效率。在压缩编码方面，MPEG-1 和 MPEG-2 的不同点在于，MPEG-1 主要以帧处理为主，而 MPEG-2 可以在场和帧两种方式中进行自适应处理。

MPEG-2 标准的制定是为了满足不断增长的对活动图像和相应音频信号进行通用编码的需要，因此，在标准制定中充分考虑到各种应用场合对分辨率和比特率等的不同要求，适用于广泛的领域。另外，不同应用又可能会有相近的功能要求，也就是要求它们能够互通。为此，MPEG-2 在编码中采用 4 种输入格式（级）和 5 类不同处理方法。须按输入信号清晰度的不同，划分为低级（LL）、主级（ML）、高 1440 级（H14L）和高级（HL）。而不同的类代表使用不同集合的码率压缩方法，分为简单类（SP）、主类（MP）、信噪比可分级类（SNRP）、空间、域可分级类（SSP）、高类（HP）。高级类对低级类具有后向兼容性。

现在 MPEG-2 标准已得到广泛应用，DVD 在视频方面即采用此标准，全国广播系统卫星传输节目采用 MPEG-2 压缩标准，在一个卫星转发器中传输 4～5 套电视节目。广播电视全国光缆网、数字微波电路、数字有线电视城域网等广播、电视信源编码均采用 MPEG-2 压缩标准。目前，各级电视台在节目制作、编辑、播出、传输中，图像压缩方式采用 MPEG-2 也已成为发展方向。在国外，美国、欧洲地面广播、电视信源编码早已采用了 MPEG-2 标准。有专家预言，MPEG-2 是今后相当一段时期国际通信、广播电视声音、图像信号压缩编码的主要标准。

MPEG-4 在经过近 6 年的研究之后于 1999 年形成国际标准，是以视频、音频、文字、数据为对象的多媒体压缩编码标准。它的编码系统是开放的，它定义的是一种格式、一种框架，而不是具体算法。它包容了 MPEG-1、MPEG-2 标准，人们还可以在系统中加入许多新的算法，使软件编、解码更方便。它采用了基于对象的编码技术，可以对场景中的不同目标进行单独编码，重要的对象采用高分辨率编码，非重要背景采用低分辨率编码，达到降低码率的同时获得更好的主观评价效果。

第7章 图像分割

在对图像的研究和应用中，人们往往仅对图像中的某些部分感兴趣，这些部分常称为目标或前景（其他部分称为背景），它们一般对应图像中特定的具有独特性质的区域。为了辨识和分析目标，需要将这些有关区域分离出来，在此基础上才有可能对目标进一步处理，如进行特征提取和测量。图像分割就是把图像空间划分成若干个具有某些一致性属性的不重叠区域并提取出感兴趣目标的技术和过程，而对图像空间的划分建立在区域的相似性和非连续性这两个概念上。

相似性就是说同一区域中的像素特征是类似的；非连续性表明不同区域间像素的特征存在突变。

图像分割是计算机视觉和图像处理中的一个基本问题，借助集合的概念可对图像分割作如下比较正式的定义。令集合 R 代表整个图像区域，对 R 的分割可以看作将 R 分成若干个满足以下四个条件的非空的子集（子区域），并且这些子区域具有以下性质：

（1） $R=\bigcup_{i=1}^{n} R_i$

（2） $R_i \bigcap R_j = \varnothing$

（3） $P(R_i)$为真

（4） $P\left(R_i \bigcup R_j\right)$为假

其中，$P(R_i)$为作用于图中所有像素的相似性逻辑程度，$i,j,=1,2,\cdots,n$。

我们可以将分割目标定位于完全分割（complete segmentation），其结果是一组唯一对应于输入图像中物体的互不相交的区域；也可以将分割目标定位于部分分割（partial segmentation），其中区域并不直接对应于图像物体。为了获得完全分割，必须与使用有关问题领域的专门知识的较高层次处理相协作。但是，有一类完全分割问题可以仅用低层处理就可以成功地解决。在这种情况下，通常图像由在均匀背景上的对比度强的物体组成，例如，简单的装配任务、血细胞、印刷字符等。这里，可以使用简单的全局方法，就可以得到将图像划分为物体和背景的完全分割。这种处理是与上下文无关的，没有使用有关物体的模型，有关分割结果的期望知识对最终的分割也没有贡献。

如果目标是部分分割，则图像被划分为分开的相对于某个选择的性质是同态的区域，选择的性质可以是亮度、色彩、反射率、纹理等。如果处理的是复杂场景的图像，例如城市场景的航拍照片，其结果也许是一组有重叠的同态区域，这样部分分割的图像必须经过进一步处理，并借助于高层信息找到最终的图像分割。

根据上述定义，我们将分割方法根据所使用的主要特征划分为三组：第一组是有关图像或图像部分的全局知识（global knowledge），这种知识一般由图像特征的直方图来表达。第二组是基于边缘的（edge-based）分割，而第三组是基于区域的（region-based ）分割，在边缘检测或区域增长中可以使用多种不同的特征。例如，亮度、纹理、速度场等。第二组和第三组解决了一个对偶问题。每个区域可以用其封闭的边界来表示，而每个封闭的边界也表达了一个区域。由于各种基于边缘和区域算法的不同性质，它们就可能给出不同的结果和由此而来的不同信息。因此这两种方法的分割结果可以结合起来构成一个单独的描述结构。

在这一章中，我们将介绍上面提出的三组方法，它们主要应用于单色图像。本书从比较简单的阈值化技术开始，介绍基于边缘的分割以及如何形成有意义的边缘，然后阐述基于区域的分割，并在最后简要介绍彩色图像的分割方法。

7.1 阈值化技术

基于阈值的分割方法是图像分割中十分古老而又简单有效的常用方法。关于这方面的文献很多。所谓阈值的方法实质是利用图像的灰度直方图信息得到用于分割的阈值。基于阈值的分割方法可以分为全局阈值的方法和局部阈值的方法。所谓全局阈值的方法是利用整幅图像的灰度信息，从整个图像中得到用于分割的阈值，并且根据该阈值对图像进行分割；而局部阈值的方法是根据图像中不同区域获得对应不同区域的几个阈值，利用这些得到的阈值对图像进行分割，也就是一个阈值对应图像中的一个子区域。

阈值分割方法也可以按照分割得到的结果分为二元阈值分割以及多元阈值分割。在二元阈值分割中，分割的结果为提取的对象以及背景；在多元阈值分割中，分割的结果为根据不同区域的特点得到的几个目标对象，所以提取每一个目标需要不同的阈值，这种方法称为多阈值方法。在实际的图像中，由于噪声等干扰因素，直方图往往不能出现明显的峰值，此时选择的阈值并不能得到满意的结果，这是由阈值分割的特点决定的。

7.1.1 灰度门限法

设图像$f(x,y)$的灰度范围属于$[Z_1,Z_2]$，根据一定的经验及知识确定一个灰度的门限，或者根据一定准则确定$[Z_1,Z_2]$的一个划分 Z_1、Z_2，其中 Z_1 代表目标，Z_2 代表背景。根据像素的灰度属于这个划分的哪个部分来将其分类，称为灰度门限法，即

如果$f(x,y)$属于 Z_1，判断(x,y)像素属于目标；

如果$f(x,y)$属于 Z_2，则判断(x,y)像素属于背景。

根据划分方法的不同，可以将灰度门限法分为：

1. 直接门限法

如果在目标区域和背景区域的内部，像素间的灰度都基本一致，而目标和背景区域的像素灰度有一定差异，可以根据灰度不同直接设定灰度门限进行分割。例如对于含有细胞的医学图像，细胞的灰度通常比背景低得多，这时可以根据某种准则求出一个门限，当像素的灰度值低于门限时，判断像素属于目标，即可将细胞提取出来。

2. 间接门限法

在有些情况下，如果对图像作一些必要的预处理然后再运用门限法，可以有效地实现图像分割。例如，图像只有黑色和白色两个灰度，但白色像素在目标区域中出现的概率比在背景区域中出现的概率大，即目标区域的平均灰度高于背景区域，可以先对图像进行邻域平均运算，再对新图运用门限法进行分割。又如，图像中目标区域灰度变化剧烈，而背景区域变化平缓，可以先对原图像进行拉氏运算，突出目标区域的特征，然后对新图使用邻域平均技

术，再用门限法实行有效地分割。

3. 多门限法

有时候一幅图像含有两个以上不同类型的区域，用直接或者间接单门限的方法无法将两个以上的目标区域提取出来，这时可以使用多个门限将这些区域划分开。对于只有两个类型区域的图像有时也要使用多门限的方法，例如图像是在照度不均条件下摄取的，若使用单一门限对整幅图像进行分割，可能发生在图像的一边能精确地把目标和背景分开，而在另一边可能把太多的背景点当作目标点保留下来的情况，或者正好相反，得不到好的分割结果。在这种情况下，可以运用同态滤波技术校正灰度，然后再用单一门限进行分割，这相当于对图像进行预处理的间接门限法。同时还可以把图像分成若干个子图，各子图像中目标和背景相对差别比较分明，可以分别对每个子图进行单门限分割，再将分割结果综合起来。

使用双门限法也可以提高对两类区域的图像的分割精度。方法是设置一高一低两个门限 t_1、t_2，不妨设 $t_1<t_2$。选择 t_2 使有些目标点的灰度大于 t_2，选择 t_1 应使任何目标点的灰度均高于 t_1。在进行分割时，把灰度大于 t_2 的像点作为“核心”目标点，对于灰度超过 t_1 的像素再根据它和核心目标点的距离判断，如果相邻则将其当作目标点。这种分割方式除了利用灰度信息还使用了空间距离信息，因此分割效果较好。

7.1.2 灰度门限的确定

分割门限选择的准确性直接影响分割的精度及图像描述分析的正确性。门限选得太高，容易把大量的目标判为背景，定得太低又会把大量的背景判为目标。因此正确分割门限是很重要的。通常采用根据先验知识确定门限，或者利用灰度直方图特征和统计判决方法确定灰度分割门限。

如果已知被处理图像的一些先验信息，可以利用试探的方法确定门限。例如已知一幅文字图像中文字占有的比例为 x,可以选择这样一个门限，使得对图像进行门限化处理后，灰度小于门限的像素的数目约占总像素的百分比为 x。

很多时候，我们并没有关于图像充分的先验知识，因此只能从图像本身的特征出发来确定门限，图像的统计特性可以提供分割的依据。利用灰度直方图特征确定灰度分割门限的原理是：如果图像所含的目标区域和背景区域大小可比，而且目标区域和背景区域在灰度上有一定的差别，那么该图像的灰度直方图会呈现双峰一谷状：其中一个峰值对应于目标中心灰度，另一个峰位对应于背景的中心灰度。由于目标边界点较少且其灰度介于它们之间，因此双峰之间的谷点对应着边界的灰度，可以将谷点的灰度作为分割门限。

由于直方图的参差性，实际上找谷值并不是一个简单的过程，需要设计一定的准则进行搜索，假设图像的直方图为 h，确定直方图谷点的位置方法之一是通过搜索找出直方图的两个局部最大值，设它们的位置是 $Z1$ 和 $Z2$，并且要求这两点距离大于某个设定的距离，然后求 $Z1$ 和 $Z2$ 之间直方图最低点 Zm，用 $h(Zm)/\min(h(Z1),h(Z2))$测度直方图的平坦性，若这个值很小，则表示直方图是双峰一谷状，可将 Zm 作为分割门限。

寻找灰度门限也可以用一个解析函数来拟合直方图双峰之间的部分，然后再用函数求极值的方法找出这个解析函数极小值的位置作为谷点，例如可用二次曲线 $y=ax^2+bx+c$ 来拟合双峰之间的直方图，求得拟合系数，则 $x=-\frac{b}{2a}$ 可作为分割门限。

类似地，也可以使用两个高斯函数拟合直方图的两个峰，然后求出两个高斯函数的交点来确定门限。

需要指出的是，由于直方图是各灰度的像素统计，未考虑图像其他方面的知识，只靠直方图分割的结果有可能是错误的。即使直方图是典型的双峰-谷特性，这个图像也未必含有和背景有反差的目标。例如一幅左边是黑色而右边是白色的图像和一幅黑白像点随机分布的图像具有相同的直方图，但是后者就不包含有意义的目标。另外，还可以利用统计判决确定门限，比如利用最小误判概率准则确定分割的最佳门限。

设图像含有目标和背景，目标点出现的概率为θ，其灰度分布密度为$p(x)$，背景点的灰度分布密度为$q(x)$。按照概率论理论，这幅图像的灰度分布密度函数为：

$$S(x)=\theta p(x)+(1-\theta)q(x)$$

假设我们根据灰度门限t对图像进行分割，并将灰度小于t的像点作为背景点，灰度大于t的像素作为目标点，于是将目标点误判为背景点的概率为：

$$\varepsilon_{12}=\int_{-\infty}^{t}p(x)\mathrm{d}x$$

把背景点误判为目标点的概率为：

$$\varepsilon_{21}=\int_{t}^{+\infty}q(x)\mathrm{d}x$$

按照总误判概率最小准则，我们选取的门限t应使下式最小：

$$\varepsilon=\theta\varepsilon_{12}+(1-\theta)\varepsilon_{21}$$

根据函数求极值的方法，对t求导令结果为零，有：

$$\theta p(t)+(1-\theta)q(t)=0$$

这是典型的统计判决方法。另外还可以采用最大后验概率方法、最小误判风险方法等。已知θ、$p(x)$及$q(x)$则可以求解出最佳门限t。对于正态分布、瑞利分布、对数正态分布，最佳门限t是容易求解的。

在不知道图像灰度分布的情况下，还可以采用模式识别中最大类间方差准则确定分割的最佳门限。其基本思想是对像素进行划分，通过使划分得到的各类之间的距离达到最大，来确定合适的门限。

设图像f中，灰度值为K的像素的数目是n_i，总像素数目是$N=\sum_{i=1}^{L}n_i$

各灰度出现的概率为：$p_i=\frac{n_i}{N}$

设以灰度k为门限将图像分为两个区域，灰度为$1\sim k$的像素和灰度为$k+1\sim L$的像素分别属于区域A和B，则区域A和B的概率分别为

$$\omega_A=\sum_{t=1}^{k}p_t \text{ 和 } \omega_B=\sum_{t=k+1}^{L}p_t$$

为了简便起见，定义$\omega_A=\omega(k)$。

区域A和B的平均灰度为：

$$\mu_A=\frac{1}{\omega_A}\sum_{i=1}^{k}ip_i\frac{\Delta\mu(k)}{\omega(k)} \text{ 和 } \mu_B=\frac{1}{\omega_B}\sum_{i=k+1}^{L}ip_i\frac{\Delta\mu-\mu(k)}{1-\omega(k)}$$

两个区域的方差为

$$\sigma^2 = \omega_A(\mu_A - \mu)^2 + \omega_B(\mu_B - \mu)^2 = \frac{[\mu\omega(k) - \mu(k)]^2}{\omega(k)[1-\omega(k)]}$$

按照最大类间方差的准则，从 1 到 L 改变 k，并计算类间方差，使上式最大的 k 就是区域分割的门限。

下面是最大方差法计算灰度分割门限的代码：

```
function th = thresh_md(a);
%  该函数实现最大方差法计算分割门限
%  输入参数为灰度图像，输出为灰度分割门限
count = imhist(a);
%  返回图像矩阵 a 各个灰度等级像素个数
[m,n] = size(a);
N = m*n - sum(sum(find(a==0),1));
L = 256;
%  计算出现概率不为 0 的最小灰度
for i=2:L
    if count(i)~=0
        st = i-1
        break
    end
end
%  计算出现概率不为 0 的最大灰度
for i=L:-1:1
    if count(i)~=0
        nd = i-1
        break
    end
end
f = count(st+1:nd+1);
%  p 和 q 分别为灰度起始和结束值
p = st;
q = nd-st;
% 计算图像的平均灰度
u = 0;
for i=1:q
    u=u+f(i)*(p+i-1);
    ua(i)=u;
end
% 计算出选择不同 k 值时，A 区域的概率
for i=1:q
```

```
    w(i)=sum(f(1:i));
end
% 求出不同 k 值时类间的方差
d = (u*w-ua).^2./(w.*(1-w));
% 求出最大方差对应的灰度级
[y,tp] = max(d);
th = tp+p;
```

图 7-1 是利用最大方差法求出灰度门限对一幅图像进行分割的结果，可以看出分割效果令人满意，细胞灰度较低，因此可以和背景区分开来。由于细胞中心灰度较亮，因此分割时被归入背景。这不是因为门限选择不正确，而是由于同一物体有不同的灰度造成的，这可以通过形态学滤波的方法来修正。

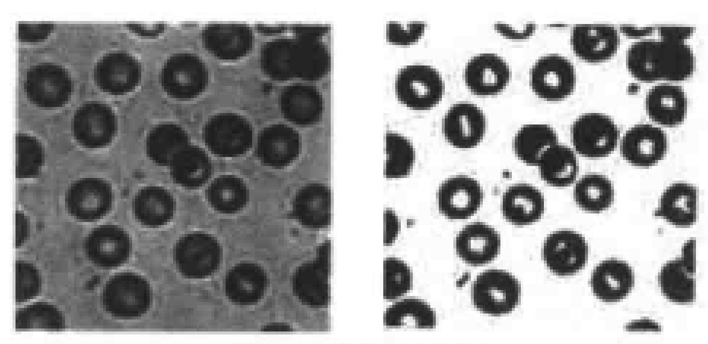


图 7-1　最大方差灰度法门

7.2　基于边缘的分割

所谓边缘是指其周围像素灰度有变化的那些像素的集合。边缘广泛存在于物体与背景之间、物体与物体之间、基元与基元之间。物体的边缘是由灰度不连续所反映的。基于边缘的分割代表了一大类基于图像边缘信息的方法，它是最早的分割方法之一，而且现在仍然是非常重要的。基于边缘的分割依赖于由边缘检测子找到的图像边缘，这些边缘标示出了图像在灰度、色彩、纹理等方面不连续的位置。在下一节我们将描述各种边缘检测算子，但是边缘检测得到的图像结果并不能用作分割结果。必须采用后续的处理将边缘合并为边缘链，使它与图像中的边界对应得更好。最终的目标是至少达到部分分割，即将局部边缘聚合到一幅图像中，使其中只出现对应于存在的物体或图像部分的边缘链。

由于噪声和模糊的存在，检测到的边界可能会变宽或在某些点处发生间断。因此，边界检测包括两个基本内容：首先抽取出反映灰度变化的边缘点，然后剔除某些边界点或填补边界间断点，并将这些边缘连接成完整的曲线。

7.2.1　边缘检测的基本原理及常用边缘检测算子

边缘检测的实质是采用某种算法来提取出图像中对象与背景间的交界线。

我们将边缘定义为图像中灰度发生急剧变化的区域边界。图像灰度的变化情况可以用图

像灰度分布的梯度来反映，因此我们可以用局部图像微分技术来获得边缘检测算子。经典的边缘检测方法是对原始图像中像素的某小邻域来构造边缘检测算子。以下是对几种经典的边缘检测算子进行理论分析，并对各自的性能特点做出了比较和评价。

不妨记：

$$\nabla f(x,y)=\frac{\partial f}{\partial x}i+\frac{\partial f}{\partial y}j$$

为图像的梯度，$\nabla f(x,y)$ 中包含局部灰度的变化信息。

记 $e(x,y)=\sqrt{f_x^2(x,y)+f_y^2(x,y)}$ 为梯度 $\nabla f(x,y)$ 的幅度，$e(x,y)$ 可以用作边缘检测算子。为了简化计算，也可以将 $e(x,y)$ 定义为偏导数 f_x,f_y 的绝对值之和：

$$e(x,y)=\left|f_x(x,y)\right|+\left|f_y(x,y)\right|$$

人们以这些理论为依据，提出了许多算法，其中比较常用的边缘检测方法有差分边缘检测、Roberts 边缘检测算子、Sobel 边缘检测算子、Prewitt 边缘检测算子、Robinson 边缘检测算子、Laplace 边缘检测算子、Canny 算子和 LOG 算子等等。

1. *差分边缘检测方法*

利用像素灰度的一阶导数算子在灰度迅速变化处得到高值来进行奇异点的检测。它在某一点的值就代表该点的“边缘强度”，可以通过对这些值设置阈值来进一步得到边缘图像。然而，用差分检测边缘必须使差分的方向与边缘方向垂直，这就需要对图像的不同方向都进行差分运算，增加了实际运算的繁琐性。

一般为垂直边缘、水平边缘、对角线边缘检测，如下所示：

$$\begin{bmatrix}0&-1&0\\0&1&0\\0&0&0\end{bmatrix}\qquad\begin{bmatrix}0&0&0\\-1&1&0\\0&0&0\end{bmatrix}\qquad\begin{bmatrix}-1&0&0\\0&1&0\\0&0&0\end{bmatrix}$$

差分边缘检测方法是最原始、最基本的方法。根据灰度迅速变化处一阶导数达到最大（阶跃边缘情况）原理，要求差分方向与边缘方向垂直，利用导数算子检测边缘。这种算子具有方向性，运算繁琐，目前很少采用。

2. *Roberts 边缘检测算子*

Roberts 边缘检测算子根据任意一对互相垂直方向上的差分可用来计算梯度的原理，采用对角线方向相邻两像素之差，即：

$$\Delta_x f=f(i,j)-f(i+1,j+1)$$
$$\Delta_y f=f(i,j+1)-f(i+1,j)$$

有了 $\Delta_x f$，$\Delta_y f$ 之后，可以根据下式计算出 Roberts 的梯度幅度值 $R(i,j)$：

$$R(i,j)=\sqrt{\Delta_x^2 f+\Delta_y^2 f}$$

或

$$R(i,j)=\left|\Delta_x f\right|+\left|\Delta_y f\right|$$

它们的卷积算子为：

$$\Delta_x f:\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\qquad \Delta_y f:\begin{bmatrix}0 & 1\\-1 & 0\end{bmatrix}$$

适当选取门限 TH，并作如下判断：如果 $R(i,j)>TH$，则(i,j)为阶跃状边缘点，$\{R(i,j)\}$为边缘图像。

Roberts 算子采用对角线方向相邻两像素之差近似梯度幅值检测边缘。检测水平和垂直边缘的效果好于斜向边缘，定位精度高，但对噪声敏感。

3. Sobel 边缘检测算子

对数字图像$\{f(i,j)\}$的每个像素，考察它上下左右邻点灰度的加权差，与之接近的邻点的权大。据此，定义 Sobel 算子如下：

$$\begin{aligned}s(i,j)&=|\Delta_x f|+|\Delta_y f|\\&=|(f(i-1,j-1)+2f(i-1,j)+f(i-1,j+1)-f(i+1,j)+2f(i+1,j)+f(i+1,j+1))|\\&\quad+|(f(i-1,j-1))+2f(i,j-1)+f(i+1,j-1))-f(i-1,j+1)+2f(i,j+1)+f(i+1,j+1)|\end{aligned}$$

卷积算子为：

$$\Delta_x f:\begin{bmatrix}-1 & 0 & 1\\-2 & 0 & 2\\-1 & 0 & 1\end{bmatrix}\qquad \Delta_y f:\begin{bmatrix}-1 & -2 & -1\\0 & 0 & 0\\1 & 0 & 1\end{bmatrix}$$

适当选取门限 TH，并作如下判断：如果 $R(i,j)>TH$，则(i,j)为阶跃状边缘点，$\{R(i,j)\}$为边缘图像。

Sobel 算子很容易在空间上实现，Sobel 边缘检测器不但产生较好的边缘检测效果，而且受噪声的影响也比较小。当使用大的邻域时，抗噪性能会更好，但这样会增加计算量，并且得出的边缘也会相应变粗。

Sobel 算子利用像素点上下、左右邻点的灰度加权算法，根据在边缘点处达到极值这一现象进行边缘的检测。Sobel 算子对噪声具有平滑作用，提供较为精确的边缘方向信息，但它同时也会检测出许多伪边缘，边缘定位精度不够高。当对精度要求不是很高时，它是一种较为常用的边缘检测方法。

4. Prewitt 边缘检测算子

Prewitt 边缘检测算子是一种边缘样板算子。这些算子样板由理想的边缘子图像构成。依次用边缘样板去检测图像，与被检测区域最为相似的样板给出最大值。用这个最大值作为算子的输出值 $P(i,j)$，这样可将边缘像素检测出来。

定义 Prewitt 边缘检测算子模板如下：

$$\begin{bmatrix}1 & 1 & 1\\1 & -2 & 1\\-1 & -1 & -1\end{bmatrix}\begin{bmatrix}1 & 1 & 1\\1 & -2 & 1\\1 & -1 & -1\end{bmatrix}\begin{bmatrix}1 & 1 & -1\\1 & -2 & -1\\1 & 1 & -1\end{bmatrix}\begin{bmatrix}1 & 1 & -1\\1 & -2 & -1\\1 & 1 & 1\end{bmatrix}$$

$$\begin{bmatrix}-1 & -1 & -1\\1 & -2 & 1\\1 & 1 & 1\end{bmatrix}\begin{bmatrix}-1 & -1 & 1\\-1 & -2 & 1\\1 & 1 & 1\end{bmatrix}\begin{bmatrix}-1 & 1 & 1\\-1 & -2 & 1\\-1 & 1 & 1\end{bmatrix}\begin{bmatrix}1 & 1 & 1\\-1 & -2 & 1\\-1 & -1 & 1\end{bmatrix}$$

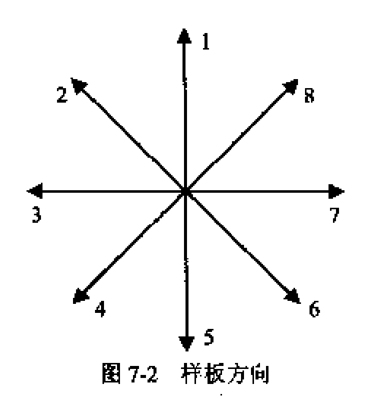


图 7-2 样板方向

8 个算子样板所对应的边缘方向如图 7-2 所示。

适当选取门限 TH，并作如下判断：如果 $P(i,j)>TH$，则(i,j)为阶跃状边缘点，$\{P(i,j)\}$为边缘图像。

5. Robinson 边缘检测算子

Robinson 边缘检测算子也是一种边缘样板算子，其算法和 Prewitt 边缘检测算子相似，只是 8 个样板不同，如图 7-3 所示。

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad \begin{bmatrix} 2 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & -1 & -2 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & -2 \\ 1 & 0 & -1 \\ 2 & -1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad \begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix} \quad \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

图 7-3 Robinson 边缘检测算子模板

6. Laplace 边缘检测算子

Laplace 边缘检测算子是一种二阶微分算子，对于数字图像$\{f(i,j)\}$,它在图像中的位置(i,j)的 Laplace 定义如下：

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Laplace 边缘检测算子是无方向性的算子，它比前面所述的多个方向导数算子的计算量要小，因为只用一个模板，且不必综合各模板的值。计算数字图像的 Laplace 值也是借助各种模板卷积实现的。实现 Laplace 运算的几种模板可见图 7-4。在数字图像中，可用差分来近似微分运算，若选用图 7-4 中的第一个检测模板，则$f(i,j)$的 Laplace 算子为：

$$\begin{aligned}\nabla^2 f(i,j) &= \Delta_x^2 f(i,j) + \Delta_y^2 f(i,j) \\ &= 4f(i,j) - f(i+1,j) - f(i,j+1) - f(i-1,j) - f(i,j-1)\end{aligned}$$

几种常用的实现 Laplace 运算的检测模板如图 7-4 所示。

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad \begin{bmatrix} -1 & 0 & -1 \\ 0 & 4 & 0 \\ -1 & 0 & -1 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

图 7-4 Laplace 运算的检测模板

由于 Laplace 算子是一种二阶导数算子，对图像中的噪声相当敏感。另外它常产生双像素宽的边缘，且也不能提供边缘方向的信息。由于以上原因，Laplace 算子很少直接用于检测边缘，而主要用于已知边缘像素后确定该像素是在图像的暗区或明区一边。

7.2.2 各种边缘检测算子的 MATLAB 实现及效果比较

在 MATLAB 中可以由 edge 函数实现各算子对边缘的检测，以 Roberts 算子为例，其语法格式如下：

BW=edge（I，'roberts'）

BW=edge（I，'roberts'，thresh）

[BW，thresh]=edge（I，'roberts'，…）

BW=edge（I，'roberts'）自动选择阈值用 Robert 算子进行边缘检测。

BW=edge（I，'roberts'，thresh）根据所指定的敏感阈值 thresh 用 Robert 算子进行边缘检测，它忽略了所有小于阈值的边缘。当 thresh 为空时，自动选择阈值。

[BW，thresh]=edge（I，'roberts'，…）返回阈值。

edge 函数对灰度图像 I 进行边缘检测，返回与 I 同样大的二值图像 BW，其中 1 表示边缘，0 表示非边缘。I 是 unit8 型、unit16 型，或者是 double 型，BW 是 unit8 型。

其余 Sobel 算子、Prewitt 算子、LOG 算子、Canny 算子的实现仅需将 'roberts' 用 'sobel'、'prewitt'、'log'、'canny' 代替即可。

下面是具体实现这几个算子的一个例程，原始图像如图 7-5 所示。

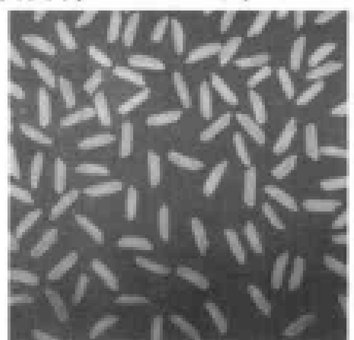


图 7-5　原始图像

```
I = imread('E:\temp\rice.bmp');
BW1 = edge(I,'sobel');    %应用 Sobel 算子进行滤波
BW2 = edge(I,'roberts');  %应用 Roberts 算子进行滤波
BW3 = edge(I,'prewitt'); %应用 Prewitt 算子进行滤波
BW4 = edge(I,'log'); %应用 LOG 算子进行滤波
BW5 = edge(I,'canny'); %应用 Canny 算子进行滤波
subplot(2,3,1),imshow(BW1)
subplot(2,3,2),imshow(BW2)
subplot(2,3,3),imshow(BW3)
subplot(2,3,4),imshow(BW4)
subplot(2,3,5),imshow(BW5)
```

进行上述各方法的边缘检测后，结果如图 7-6 所示。

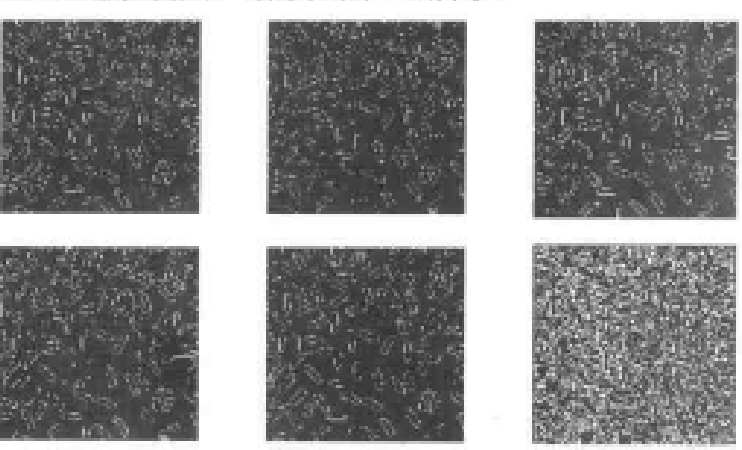


图 7-6　使用各种边缘检测算子得到的边缘图像

7.2.3 直线提取

因为直线通常对应重要的边缘信息，直线提取是计算机视觉中一项非常重要的技术。例如车辆自动驾驶技术中道路的提取需要有效地提取直的道路边缘，也就是提取获取的图像中的直线；而航空照片分析中，直线更是对应于重要的人造目标的边缘。因此把直线单独提取出来进行研究很有意义。况且，由于直线有其不同于一般曲线的特征，因此它的提取方法也与一般的边缘检测方法不同。

1. Hough 变换法

利用 Hough 变换法提取直线是一种变换域提取一直线的方法，它把直线上点的坐标变换到过点的直线的系数域，巧妙地利用了共线和直线相交的关系，使直线的提取问题转化为计数问题。Hough 变换提取直线的主要优点是受直线中的间隙和噪声影响较小。

一般常将 Hough 变换称为线—点变换，这是因为它将直角坐标系中的线变为极坐标系中的点，如图 7-7 所示。图中（a）所示直线可以用参数表示为：

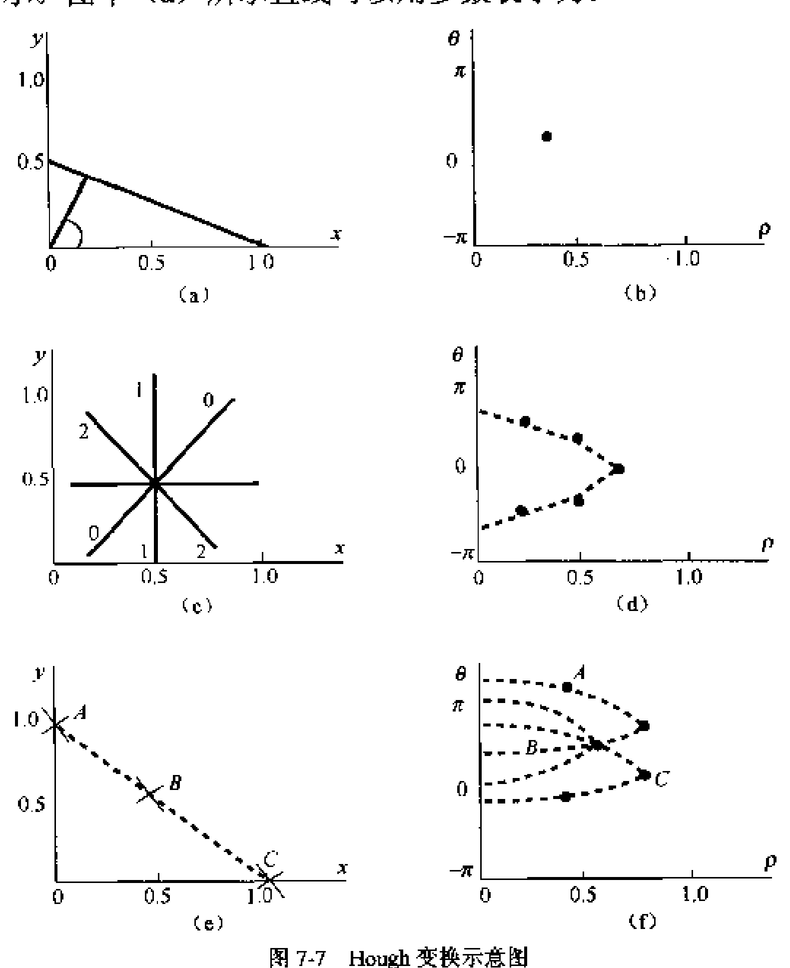


图 7-7　Hough 变换示意图

$$\rho = x\cos\theta + y\sin\theta$$

其中ρ为从原点到直线的垂直距离，θ为从 x 轴算起的角度，这条直线在$\rho-\theta$平面中为一点，见图（b)。而通过 x–y 平面上一点的一簇直线变换到$\rho-\theta$平面时，将形成一条类似正弦状的轨迹，见图（c）和（d)，也即 x–y 平面上一点对应$\rho-\theta$平面上这样一条曲线。若在 x–y 平面上有三个共线点，它们变换到$\rho-\theta$平面上为有一公共交点的三条曲线，交点的$\rho-\theta$参

数就是三点共线直线之参数。根据这个原理，可以用 Hough 变换抽取直线。通常将 x–y 称为图像平面，$\rho-\theta$ 称为参数平面。

抽取直线的 Hough 变换可以概括如下：

（1）在 σ、θ 合适的最大值和最小值之间建立一个离散的参数空间；

（2）建立一个累加器 $A(\sigma,\theta)$，并置每个元素为 0。

（3）对梯度图上超过门限值的每一点作 Hough 变换，即算出该点在 $\sigma-\theta$ 网格上的对应曲线，并在相应的累加器加 1，即

$$A(\rho,\theta)=A(\rho,\theta)+1$$

（4）找出对应图像平面共线点的累加器上的局部极大值，这个值就提供了图像平面上共线点共线的参数。

Hough 变换的基本策略是：由图像空间中的边缘数据点去计算参数空间中的参考点的可能轨迹，并在一个累加器中给计算出的参考点计数，最后选出峰值。

【应用实例】

在某些制导可见光探测的应用中，一方面在目标距离探测器比较远的情况下，弱运动目标可以认为是点目标。另一方面在某些制导方式下导弹运动时，由于弹体的抖动，在探测器曝光时间内可认为远处的弱运动目标成像为一条直线段。假设探测器曝光时间为 20ms，抖动幅度为 20 度/s，视场为 3 度，弱运动的点目标在 256×256 像素的图像中成像为长约 34 实际像素长的线段。此时弱运动目标的检测可以转换为对图像中弱线段的检测。但是由于可见光探测器相对红外探测器而言曝光时间比较长，同时考虑导弹的高速运动，当前帧和下一帧图像的特性差别较大，所以红外图像目标检测中常用的序列图像多帧累加处理的方法在此应用中不适用。基于以上分析，检测目标只能在单帧低信噪比图像中进行，这里我们选择应用 Hough 变换来实现对弱线段的检测。

下面例程中，首先在噪声服从高斯分布的背景中构造信噪比为 2、长度为 34 像素的目标线段（这里假设图像大小为 128×128）；在找出线段所在直线后，在采用固定长线段滑动的方法，确定线段的起始点，最终确定线段的确切位置。

```
loop_l=1 %设定循环检测次数，这里设为只循环一次
tip_l=1 %设定 hough 变换的遍历步径
xl=30   %设定线段起始点
xr=60   %设定线段终止点
L=xr-xl
M=zeros(1,loop_l)
B=zeros(1,loop_l)
X=zeros(1,loop_l)
for j=1:loop_l
    colordef white
    im_in=randn(128,128); %构造服从高斯分布的噪声背景，图形大小为%128*128
    orig_m1=1
    orig_b1=30
    for x=1:128;
        y=floor(orig_m1*x + orig_b1);
        if(x <= xr & x >=xl)
```

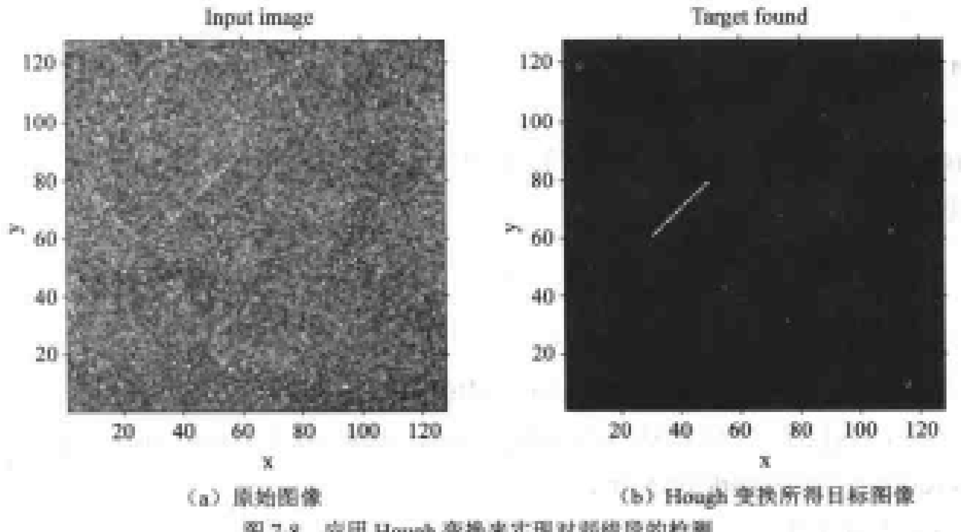
```
            im_in(y,x)=im_in(y,x)+2;
        end;
    end;
    im=wiener2(im_in,[3,3]); %wiener filter
    for thet=1:1:360;
        sinarray(thet)=sin(thet*pi/180.0);
        cosarray(thet)=cos(thet*pi/180.0);
    end;
    A=zeros(181,360);
    for x=1:128;
        for y=1:128;
            for theta=1:360;
                r=round(x*cosarray(theta) + y*sinarray(theta));
                if r<1
                    r=1;
                    A(r,theta)=A(r,theta);
                else
                    A(r,theta)=A(r,theta)+im(y,x);
                end;
            end;
        end;
    end;
    A(r,theta);
    m=max(max(A));
    for r=1:181;
        for theta=1:360;
            if A(r,theta)==m
                maxr = r;
                maxtheta = theta;
            end;
        end;
    end;
    m=-1.0*((cos(maxtheta*pi/180.0))/(sin(maxtheta*pi/180.0)));
    b=maxr/(sin(maxtheta*pi/180.0));
    Z=zeros(1,100)
    for x=1:tip_l:128
        if x<(128-L)
            z=floor(x/tip_l)+1
            for i=0:L
                k=x+i;
```

```
                        y=round(m*(k)+b);
                        if (y <= 127 & y >=2)
                            Z(1,z)=Z(1,z)+im(y,k);
                        end;
                    end;
                end;
            end;
            maxz=max(max(Z));
            p=1;
            my_x=1;
            for z=1:100;
                if Z(1,z)==maxz
                    p=z
                    my_x=p*tip_l;
                end;
            end;
            im2=zeros(128,128);
            for x=my_x:1:my_x+L;
                y=round(m*x + b);
                if(y <= 128 & y >=1)
                    im2(y,x)=255;
                end;
            end;
            M(1,j)=m
            B(1,j)=b
            X(1,j)=my_x
        end;
        jj_t=0;
        jj_f=0;
        for ii=1:loop_l
            if
        (((orig_m1-0.15)<=M(1,ii))&(M(1,ii)<=(orig_m1+0.15))&((orig_b1-2)<=B(1,ii))
&(B(1,ii)<=(orig_b1+2)))
                jj_t=jj_t+1
            end
            center_x=round(X(1,ii)+L/2)
            center_y=round(M(1,ii)*center_x+B(1,ii))
            if ( ((xl-3)<=X(1,ii))&(X(1,ii)>=xl+3) ) &((center_y>=round( orig_m1*(xl+L/2) +
orig_b1)-3)&(center_y<=round( orig_m1*(xl+L/2) + orig_b1)+3))
            jj_f=jj_f+1
```

```
        end
    end
    m_av=sum(M)/loop_l
    b_av=sum(B)/loop_l
    my_x_av=sum(X)/loop_l
    subplot(121), image(im_in)
    title('Input image')
    axis equal
    axis([1 128 1 128])
    axis xy
    xlabel('x')
    ylabel('y')
    title('Input image')
    im2=zeros(128,128);
    for x=round(my_x_av):1:round(my_x_av)+18;
        y=round(m_av*x + b_av);
        if(y <= 128 & y >=1)
            im2(y,x)=255;
        end;
    end;
    subplot(122), image(im2 )
    axis equal
    axis([1 128 1 128])
    axis xy
    xlabel('x')
    ylabel('y')
    title('Target found')
```



图 7-8 是原始图像和经 Hough 变换后检测出的目标图像。



（a）原始图像　　（b）Hough 变换所得目标图像

图 7-8　应用 Hough 变换来实现对弱线段的检测

由图可以看出 Hough 变换可以很好地检测出被淹没在噪声中的直线段。但是 Hough 变换的缺点是运算量太大，而且由于不考虑各点之间的距离信息，因此容易将不属于直线的点也连接到直线上，即产生所谓的过连接现象。实际中，为了克服运算量大的缺点，通常可以把图像划分为小块，对各块图像利用 Hough 变换提取直线，然后再将各直线连接起来，或者是增加变换过程中的遍历步径，来减少运算量。

2. 相位编组法

前面提到的边缘提取算法，比如梯度模算子法，拉普拉斯高斯算子法等都有一个共同的特点，即利用的是梯度的幅度信息。如果某一点的梯度模超过了预先设定的门限，则认为存在边缘。它的基本原理是，认为边缘是灰度发生突变的地方。在图像的灰度缓慢变化的情况下，利用这些方法效果不是很好。

梯度相位法采用的是另一种思路，它认为边缘不只是在灰度发生突变的地方存在，如果某个区域，灰度沿某个方向缓慢变化，这个区域也存在边缘，只不过认为边缘比较粗。

因此，梯度的方向也是非常重要的信息，如果在某个区域，各点的梯度的方向相同或者相近，则这个区域很可能存在边缘。相位编组法的思想是求出图像中各个像素的梯度相位，将相邻的方向相同的点编为一个直线支撑区，然后根据支撑区内像素的坐标求出直线的方程，即可将直线提取出来。图 7-9 中箭头代表梯度方向，相同方向的箭头被编为一组，构成直线支撑区。

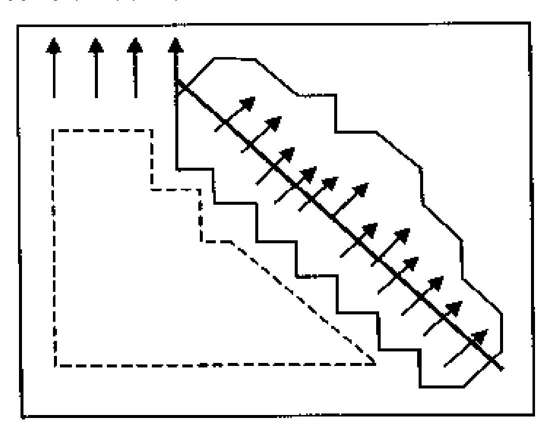


图 7-9 相位编组原理示意图

利用相位编组法提取直线可以按照下面的步骤进行：

（1）将梯度方向相似的、相邻近的边缘编成边缘支持区域，显然这个区域不受具体尺寸的限制。

（2）由加权平面近似灰度表面，用有关像元的梯度幅度值对拟合加权，以使边缘最陡的部分起主导作用。

（3）从边缘支持区域和拟合平面提取属性，这些属性包括代表此区域的直线及其长度、对比度、宽度、位置、方向和直线度等。

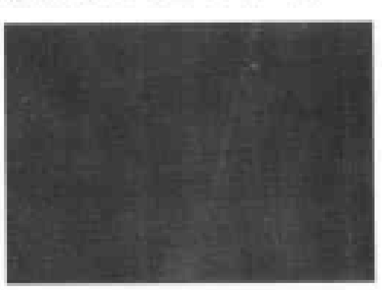
（4）根据属性计算得到各种不同的图像事件，诸如长直线、高对比度直线（强纹理）、低对比度短直线(弱纹理)以及在特定方向和位置上的直线。

```
function gradephase(x)
[m,n]=size(x);
bw=edge(x,'sobel');
gy=x(1:m-1,1:n-1)-x(2:m,1:n-1);
gx= x(1:m-1,1:n-1)-x(1:m-1,2:n);
g=gy./(gx+eps);
ph-atan(g)+(sign(gx)<0&sign(gy)>0)*pi+…
        (sign(gx)<0&sign(gy)>0)+ (sign(gx)>0&sign(gy)<0)*2*pi+(sign(gx)==0)*pi;
grdgp=floor(ph/pi*4);
cn=0;s=[ ];
```

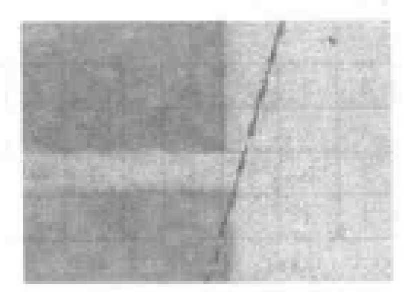
```
In_spt=cell(1,1);
%cell 数组类似 c 语言的指针，可以动态改变大小。S 用来存放直线编组区域内的像%素坐标。
Pline=[];
for  i=2:m-1
for  j=2:n-1
    if  bw(i,j)==D&gradgp(i,j)==0 %搜索边缘点。
        ph_cp=gradgp(i,j); gradgp(i,j)=0; bw(i,j)=0;
        cn=cn+1;p=[i,j];
        s=[s,p];in_spt(cn)-[p];
        while ～isempty(s) %为了在 8 个邻域进行搜索，采用入栈和出栈操作。
            [cs,rs]=size（s）;
            ps=s(:,rs);s=s(:,1:rs-1);
            col=ps(1,1);row=ps(2,1);
            if  legal(col+1,row,m,n)&grdgp(col+1,row)= =ph_cp
                s= [s,[col+1,row]']; in_spt(cn)=[ in_spt(cn), [col+1,row]'];
                bw(col+1,row)=0;gradgp(col+1,row)=0;
            end
            if  legal(col+1,row+1,m,n)&grdgp(col+1,row+1)= =ph_cp
                s= [s,[col+1,row+1]']; in_spt(cn)=[ in_spt(cn), [col+1,row+1]'];
                bw(col+1,row+1)=0;gradgp(col+1,row+1)=0;
            end
            if  legal(col,row+1,m,n)&grdgp(col,row+1)= =ph_cp
                s= [s,[col,row+1]']; in_spt(cn)=[ in_spt(cn), [col,row+1]'];
                bw(col,row+1)=0;gradgp(col,row+1)=0;
            end
            if  legal(col-1,row+1,m,n)&grdgp(col-1,row+1)= =ph_cp
                s= [s,[col-1,row+1]']; in_spt(cn)=[ in_spt(cn), [col-1,row+1]'];
                bw(col-1,row+1)=0;gradgp(col-1,row+1)=0;
            end
            if  legal(col-1,row+1,m,n)&grdgp(col-1,row+1)= =ph_cp
                s= [s,[col-1,row+1]']; in_spt(cn)=[ in_spt(cn), [col-1,row+1]'];
                bw(col-1,row+1)=0;gradgp(col-1,row+1)=0;
            end
            if  legal(col-1,row,m,n)&grdgp(col-1,row)= =ph_cp
                s= [s,[col-1,row]']; in_spt(cn)=[ in_spt(cn), [col-1,row]'];
                bw(col-1,row)=0;gradgp(col-1,row)=0;
            end
            if  legal(col,row-1,m,n)&grdgp(col,row-1)= =ph_cp
                s= [s,[col,row-1]']; in_spt(cn)=[ in_spt(cn), [col,row-1]'];
                bw(col,row-1)=0;gradgp(col,row-1)=0;
```

```
            end
            if   legal(col-1,row-1,m,n)&grdgp(col-1,row-1)= =ph_cp
                 s= [s,[col+1,row-1]']; in_spt(cn)=[ in_spt(cn), [col+1,row-1]'];
                 bw(col+1,row-1)=0;gradgp(col+1,row-1)=0;
            end
            %以上判断是在 8 个方向搜索相同方向的像素。
            If    length(in_spt{cn})<=10
                  In_spt{cn}=[];
                  cn=cn-1;
                 %滤除短线段
            else
                 plane=draw_l(in_spt{cn});
                 %实现画线功能，代码与 Hough 变换中的代码相同。
                 pline=[pline,plne];
                 c(i)=length(in_spt{cn});
            end
         end
      end
   end
end
```

图 7-10 是一幅利用相位编组法提取直线得到的结果，可以看出，利用相位编组法提取直线，可以将灰度缓慢变化的边缘提取出来。



（a）原始焊缝图像



（b）相位编组法提取直线的结果

图 7-10　利用相位编组法提取直线

7.3　基于区域的分割

本节讨论一类应用区域生长、区域分裂与区域合并技术的图像分割算法，这些算法又简称为区域生长法。

7.3.1　区域生长的基本概念

在前面所讨论的边缘提取和灰度域值法中，我们感兴趣的是像素值的差别，而对于研究

区域生长来说则是寻找具有相似性的像素群，它们对应某种实体世界的平面或物体。区域增长的基本思想是将具有相似性质的像素集合起来构成区域。具体先对每个需要分割的区域找一个种子像素作为增长的起点，然后将种子像素周围邻域中与种子像素有相同或相似性质的像素合并到种子像素所在的区域。将这些新像素当作新的种子像素继续进行上面的过程，直到再没有满足条件的像素。图 7-11 说明了区域生长的基本目的。区域生长的一种最简单的方法是从某个像素开始，然后检查它的近邻，判断它们是否有相似性，这个相似性准则可以是灰度级、彩色、组织、梯度或其他特性。例如，如果它们有相似的亮度，那么，就将它们合起来以形成一个区域。这样，就从单个像素往外生长出区域。

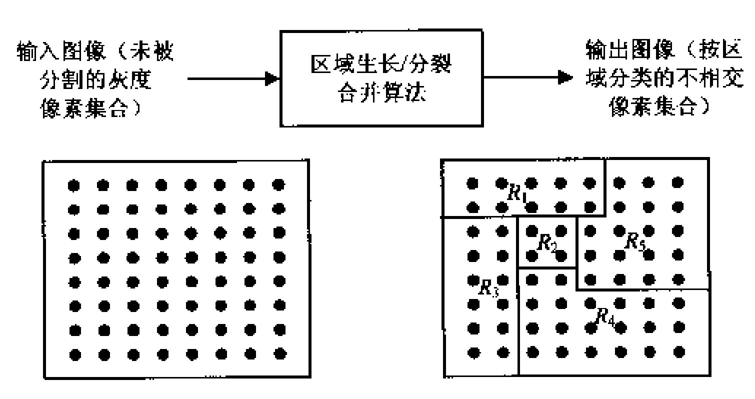
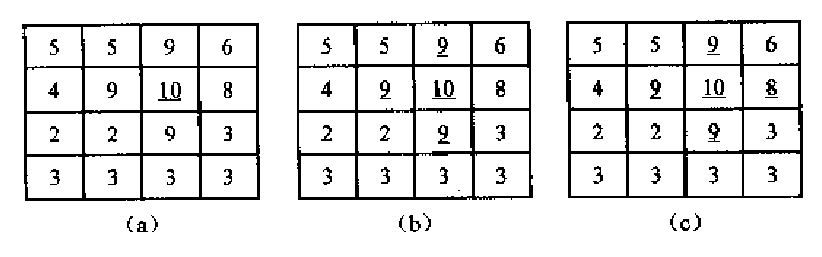


图 7-11　区域生长

图 7-12 示出了一个简单的区域生长的例子。这个例子的相似性准则是邻近点的灰度级与物体的平均灰度级的差小于 2。图中被接受的点和起始点均用一短线标出，其中图（a）是输入图像；图（b）是第一步接受的邻近点；图（c）是第二步接受的邻近点。这种区域生长方法是一个自底向上的运算过程。



(a)

5	5	9	6
4	9	10	8
2	2	9	3
3	3	3	3

(b)

5	5	9	6
4	9	10	8
2	2	9	3
3	3	3	3

(c)

5	5	9	6
4	9	10	8
2	2	9	3
3	3	3	3

图 7-12　区域生长简例

当生长任意物体时，相似性准则可以不是像素的亮度，而可以采用所谓结构判断准则。因此，此时的区域生长不是从像素开始而是将一幅图像分成一组小的区域。然后对每个区域应用均匀性检查，如果这一检查不成功，那么，对该区域再进行分裂操作。重复进行这一过程直到所有的区域是均匀时为止。这样，区域是由较小的区域而不是由像素生长的。显然这是一个自顶向下的运算过程。应用区域分裂/合并技术来实现区域生长存在两个关键问题：一是选择区域分裂的方法，二是分裂中止的判断准则。这种方法的具体实现和用于表示图像的数据结构关系很密切。采用小的区域而不用像素的主要优点是降低了对噪声的灵敏度。

值得注意的是，区域生长技术的计算复杂费时，因此很少应用于对实时要求高的场合。只有当阈值分割或边缘提取技术无法产生满意的结果时，才考虑应用区域生长方法。

7.3.2 用平均灰度分割

一个像素的值和一个区域上的平均值之间的最大差值可以作为一种均匀性判断准则。设 $f(p)$为图像像素灰度值，对大小为 N 的区域 R，令

$$m=\frac{1}{N}\sum_{P\in R}f(P) \tag{7.1}$$

于是，如果对某一阈值 T，区域 R 内的像素灰度值满足

$$\max_{P\in R}|f(P)-m|<T \tag{7.2}$$

则称该区域是均匀的，这种均匀的判断是试探性的，并且可以把它作为其他均匀性准则的模型。下面说明这种方法的理论依据。

假设要处理的是具有零均值的高斯白噪声图像，这就意味着在像素 P 上，噪声值为 l 的概率由下式决定：

$$p_x(l)=\frac{1}{\sqrt{2\pi}\sigma}\mathrm{e}^{-l^2/2\sigma^2} \tag{7.3}$$

式中σ是噪声的标准偏差。由于是白噪声，所以 $p_x(l)$的值与像素点的位置完全无关。实际上，它表示噪声以同样的方式影响所有的像素。像素 P 的灰度值与其平均值的差大于某个量 x 的概率由式（7.4）积分给出

$$g_m(x)=\frac{2}{\sqrt{2\pi}\sigma}\int_x^{+\infty}\mathrm{e}^{-l^2/2\sigma^2}\mathrm{d}l \tag{7.4}$$

式中的系数 2 是由于考虑到自平均的偏离有正有负，上式右边称为误差函数 erf(t)，表 7-1 列出了几个典型值。

表 7-1　　误 差 函 数

t	1.0	1.5	2.0	2.5	3.0	3.5	4.0
erf(t)	0.317	0.134	0.046	0.012	0.003	0.0005	0.0003

如果一个区域是均匀的，可以证明式（7.4）是该区域灰度值的最佳估值器。在某种情况下，像素值偏离仅仅是由于噪声引起的。因此，如果选 $l=T$，那么对某个像素，式（7.2）不满足的概率将由式（7.4）所确定。更确切地说，它等于 erf(T/a)。例如，使式（7.2）的 T=2a，那么对一特定像素来说，这个条件不满足的概率为 4.6%，而对于 T=3a，这一概率约为 0.3%。我们用 $p(T)$表示这一概率的值。于是，每个像素满足式（7.2）的概率为 1−$p(T)$，由于 N 是每个区域的像素数，因此不能成功地识别一均匀区域的概率仅为$1-[1-p(T)]^N$。若 p(T)值比 l/N 小得多，这个量就近似等于 $Np(T)$。选阈值 T 等于噪声标准偏差的 3 倍，并假定对于由 256 个像素构成的 16×16 方形区域作均匀性检查，可计算求得不满足的概率等于 54%。如果阈值是标准偏差的 4 倍，那么所求得的同一概率仅为 2.5%，从实际的观点来看这是一个可以接受的值。

将均匀区域误为非均匀区域不是唯一可能的误差，还必须估计将非均匀区域误为均匀区域的概率。在这种情况下，m 和像素值的差也是由于图像各区域的值之间的差所造成的。令 m_1和 m_2是这些值，且令一个区域上 q_i%的像素是其真值并为 $m_i(i=1,2)$ 的像素。如果该区域大到在估计均值时足以忽略噪声的影响，那么均值将是 $q_1m_1+q_2m_2$。如果一个像素有真值 m_1，

那么这个值与此估计的平均值之间的差就为

$$\Delta m = m_1 - (q_1 m_1 + q_2 m_2) \tag{7.5}$$

因此，使观测值与 m_1 相差 $T + \Delta m$ 或 $T - \Delta m$，就能出现这个值与 $q_1 m_1 + q_2 m_2$ 的差大于 T 的情况，每种情况发生的概率为

$$p_i = \frac{1}{2}[P(|T - \Delta m|) + P(|T + \Delta m|)] \tag{7.6}$$

因此，p_i 是对其真值为 m_i 的像素的观察值不满足方程 7.2 的概率。设有任何一个像素不满足该方程的概率为

$$p_z = (1 - p_1)^{q_1 n}(1 - p_2)^{q_2 m} \tag{7.7}$$

式中的 p_z 是用类似于 p_1 的方式定义的。这样，p_n 是把一个实际上并不是均匀的区域当作是均匀的区域的概率。显然，如果 Δm 与 T 相比很小，那么 p_1 就接近于 $p(T)$。如果这对 p_2 也成立，那么 p_1 就近似等于 $[1 - p(T)]^N$，它与误将非均匀的区域当作为均匀区域的概率是相同的。换句话说，均匀性的检测好像是一个随机事件，当一个区域包含的像素几乎属于同一类时，就出现 Δm 值最小的情况，因此，将它当作为均匀区域并不会引起较大误差。另一方面，如果一个区域是近似相等的两种像素类型的混合，即若 q_1=q_2 使

$$\Delta m = \frac{1}{2}[m_1 \quad m_2] \tag{7.8}$$

则我们希望以很低的概率称该区域为均匀的。如果差值 Δm 的绝对值比噪声的标准偏差大得多，这种情况就会发生，因为此时 Δm 可与 T 的大小相比拟。在这些条件下，式（7.6）的方括号中第一项的自变量将接近于零，所以对应的概率将接近于 1，第二项中的自变量为标准偏差的很多倍，因此对应的概率接近为零。由这些假设可得 p_1 近似等于 0.5。也可求得 p_2 为相同的值。于是 p_n 近似等于 0.5^N。在这种情况下，不满足方程则意味着它是一个非均匀区域，其概率为 $1-0.5^N$。对 N=256，这一概率非常接近于 1。

7.3.3 基于相似统计特性的分割

上一小节的分析表明，把式（7.2）的均匀性准则用在将一个区域当作为非均匀区域方面可能会导致错误。例如当一幅图像用类似于式（7.2）的准则被分割时，常常会出现有大量的小区域似乎在图像中并没有任何真实的对应物。而利用相似统计特性来寻找具有均匀性的区域可以避免出现这些问题。这种方法是通过将一个区域上的统计特性与在该区域的各部分上所计算出的统计特性进行比较来判断区域的均匀性。如果它们相互很接近，那么这个区域可能是均匀的。这种方法对于纹理分割来说是有用的。我们可以在每一群区域上计算共生矩阵，然后比较这些矩阵，如果它们是相似的，那么这些区域的和是一个均匀区域。通常，令为区域 R 上计算出的一个特征，如果 R_{12} 是两个相邻但不相连的区域 $F(R_1)$ 与 $F(R_2)$，则可通过使 $F(R_{12})$ 接近于 $F(R_1)$ 和 $F(R_2)$ 来定义均匀性准则，R_1 可能是早已求得的区域，R_2 是被考虑要加到 R_1 上的小区域。必须选一个阈值 T，这样当 $F(R_1)$ 和 $F(R_2)$ 之差的绝对值低于 T 时，我们就认为它们是均匀的。这里 T 必须大于由噪声引起的 $F(R)$ 的方差。但是，在很多情况下，这一方差比噪声本身的方差小得多，而由于不均匀性所引起的方差却具有和以前相同的大小。因此，基于统计特征比较的均匀性准则比式（7.2）所给出的准则更为准确。

7.4 彩色图像分割

彩色图像分割是数字图像处理领域一类非常重要的图像分析技术，在对图像的研究和应用中，根据不同领域的不同需要，在某一领域往往仅对原始图像中的某些部分感兴趣。这些目标区域一般来说都具备其自身特定的一些诸如颜色、纹理等性质，彩色图像分割主要根据图像在各个区域的不同特性，而对其进行边界或区域上的分割，并从中提取出所关心的目标。

图像分割注重对图像中的目标进行检测与测量，这与在像素级对图像进行操作的图像处理技术，为改善图像视觉效果而强调在图像之间所进行的变换是有所区别的。通过对图像的分割、目标特征的提取，可将经初步图像处理的图像特征向量提取出来，并将原始的数字图像转化成为一种有利于目标表达的更抽象、更紧凑的表现形式，从而使高层的图像分析、图像理解以及计算机的模式自动识别成为可能。多年来，彩色图像分割技术一直在工业自动化控制、遥感遥测、微生物工程以及合成孔径雷达（SAR）成像等多种工程应用领域得到相当广泛的应用。

7.4.1 色彩空间

表达颜色的色彩空间有许多种，它们常是根据不同的应用目的而提出的。下面将围绕彩色图像分割，介绍几种常用的色彩空间和它们的特点。

最常见的色彩空间是红绿蓝（red,green,blue,RGB）空间，它是一种矩形直角空间结构的模型，是通过对颜色进行加运算完成颜色综合的彩色系统。它用 R、G、B 三个基本分量的值来表示颜色，它是面向硬件设备的（如 CRT），物理意义明确但缺乏直感。与它对应的是深蓝、品红和黄的 CMY 空间，主要用于非发射式显示，如彩色打印机、绘画等。

通过对不同类型图像的分析，有人经过大量试验提出可用由 R、G、B 经过线性变换得到的三个正交彩色特征

$$I_1=(R+G+B)/3$$
$$I_2=(R-B)/2 \text{ 或 } I_2=(B-R)/2$$
$$I_3=(2G-R-B)/4$$

来进行分割。这三个特征中，I_1 是最佳特征，I_2 是次佳特征，只用 I_1 和 I_2 作特征对大多数图像已可得到较好的分割效果。

彩色图像常用 R、G、B 三分量的值来表示，但 R、G、B 三分量之间常有很高的相关性，直接利用这些分量常常不能得到所需的效果。为了降低彩色特征空间中各个特征分量之间的相关性，以及为了使所选的特征空间更方便于彩色图像分割方法的具体应用，实际中常需要将 RGB 图像变换到其他的彩色特征空间中去。

比较接近人对颜色视觉感知的是色度、饱和度和亮度(hue,saturation,intensity,HSI)空间。其中 I 表示颜色的明暗程度，也有用 V(value)表示的，主要受光源强弱影响，H 表示不同颜色，如黄、红、绿，而 S 表示颜色的深浅。注意 HSI 模型有两个重要的事实作为基础，首先，I 分量与彩色信息无关，其次 H 和 S 分量与人感受彩色的方式紧密相连。HSI 空间比较直观并且符合人的视觉特性，这些特点使 HSI 模型非常适合基于人的视觉系统对彩色感知特性的图

像处理。从 RGB 到 HSI 的转换关系为：

$$H=\arccos((2R-G-B)/(2\mathrm{sqrt}(\mathrm{sqr}(R-G)+(R-B)(G-B))))$$

$$B>G \quad H=2Pi-H$$

$$S=1-3\min(R,G,B)/(R+G+B)$$

$$I=(R+G+B)/3$$

其中 S 也有用下式计算的：$S=\max(R,G,B)-\min(R,G,B)$

另外，孟塞尔色空间也用一个三维空间的模型将各种表面色的三种视觉特性：亮度、色度、饱和度全部表示出来。孟塞尔颜色系统的颜色样品在视觉上是均匀的，因而可以用它来考察和验证与某一色差公式有关的颜色空间的均匀性。孟塞尔的色度值、亮度值、彩度值大致反映了物体颜色的心理规律，代表了颜色的色度、亮度和饱和度的主观特性，但孟塞尔空间完全以主观色表为基础，没有数学表达式，使用起来很不方便。除了这些常用的颜色系统外，还有如 YIQ、YUV、YCbCr 等色彩系统，本文就不再说明了。

7.4.2 彩色分割方法

彩色图像分割是图像处理中的一个主要问题，也是计算机视觉领域低层次视觉中的主要问题。

总的来说，彩色图像分割的方法可以分为基于像元、区域、边缘的分割这三大类，前两类利用的是相似性，基于边缘的分割则是利用的不连续性。

1. 基于像元的分割方法

基于像元的分割方法又可分为三类：直方图门限技术、色彩空间聚类法以及模糊聚类分割方法。其中直方图门限技术是最常用的，由于图像门限处理的直观性和易于实现的性质，使它在彩色图像分割应用中处于中心地位。

直方图门限技术：Tominaga 提出可将 RGB 色彩空间转换成 HVC 或其他色彩空间，如 HSI，再分别求 H、V、C（或 H、S、I）的一维直方图，寻找最明显的峰值，一般是选定两个作为门限。Holla 将 RGB 色彩空间转换成 RG、YB、I，再将这三个通道用带通滤波器平滑，滤波器中心频率过滤这三种色彩特征的比率是 I:RG：YB=4:2:1，然后在二维直方图 RG-YB 中寻找峰值点和基点，从而将像素点分为两个区域。但是该方法会在图像中留下捕捉不到的部分，因此可以再考虑其他的特征进去，如亮度或者像素的局部相连性，这样可以增强分割效果。Stein 的方法是对 Holl 的改进。算法中加入了邻域的特征。当留下了一些没有被分配到的像素点时，就取它周围的 3×3 的模板，如果模板中有一个或者多个像素点被指派到区域 A，则该像素点也被指派到同样的区域 A 中去。如果该邻域模板中的像素点也没有被指派到任何区域或者被指派到了不同的区域，那么该像素点仍然不被指派。这样的话可能还是有残留点，但是比率要少得多。R.Ohtander 的方法是比较经典的，它采用九个色彩特征：R、G、B、H、S、V、Y、I、Q，对这九个特征分别计算直方图，再选择最好的峰值作为门限。Ohta 等提出的方法和前面不同点在于它将 RGB 色彩空间转换为另外定义的 I1、I2、I3 特征，再分别对它们进行直方图化，以三个一维直方图上可以看到各自的峰值点，该算法给出的 I1、I2、I3 的表达式相当于动态 K-L 变换的结果，而且都是对 R、G、B 的线性变换，不存在奇异点，不同的图像对 I1、I2、I3 各自的峰值点分割的效果有差别，需要自动选取合适的门限。

根据 I1、I2、I3 的直方图，有明显双峰的更适合该图像。

色彩空间聚类法：该方法也结合了直方图阈值选取技术。先将 RGB 色彩空间转换成 HLS 色彩空间（H、L、S 的表达式已给出），根据 L 的值将图像分为过亮区域和非过亮区域，在过亮区域里以 H 为主要特征，根据直方图取峰值进行分割，在非过亮区域里以 S 为主要特征，根据直方图取峰值进行分割，最后将分割的两副图像合并。Ferri 则是通过神经网络将像素分成几个区域，再利用编辑和压缩技术来减少分类的个数。该方法用的是 YUV 色彩空间，它对每个像素点（i,j）扩展成矢量 $F(i,j)=\{U(i,j), V(i,j), U(i+h,j), V(i+h,j), U(i-h,j), V(i-h,j), U(i,j+h), V(i,j+h), U(i,j-h), V(i,j+h)\}$，其中 h 是期望分割的目标的大小。Lauterbach 是在 LUV 色彩空间中进行分割的，首先求二维 UV 直方图的最高点，这个最高点是通过计算累计直方图的值和一个邻域窗的均值之差得到的。然后添加色彩匹配线（acl），这条线是通过两个聚类中心的一根直线。像素值在 UV 空间的那两个聚类中心之间的 acl 的欧氏距离决定了像素点被分派到哪两个类中间去。最后再在两类中用最小距离准则找一类。但是，该方法没有考虑亮度，所以在某些情况下不太适用。

模糊聚类分割方法，基于门限和模糊 C-均值法：先粗糙地用标量空间分析的一维直方图分割。具体步骤：计算图像每一个色彩特征的直方图；标量分析直方图；定义合法的几个类 $V1$，$V2$，…，Vc；对属于类别 Vi 的每一个像素点 p，用 i 标记 p；计算每一类 Vi 的重心；对没有被分类的像素值 $p(x, y)$，用模糊成员函数 U 计算，取最大的 $U(x, y)$（此时类别为 Vk），则将该像素 p 分派到 Vk 类。

2. *基于区域的分割方法*

由于彩色图像分割的目的是将图像划分为不同区域。基于像素的分割是通过用以像素性质的分布为基础的门限来进行的，比如灰度级的值或颜色，在这一节里讨论的是以直接找寻区域为基础的分割方法，主要可分为区域生长和区域分离与合并两类技术。

区域生长：区域生长是一种根据事前定义的准则将像素或子区域聚合成更大区域的过程。基本的方法是以一组“种子”点开始将与种子性质相似（诸如灰度级或颜色的特定范围）的相邻像素附加到生长区域的每个种子上。不同的方法相似性准则不一样，该准则的选择不仅取决于面对的问题，还取决于有效图像数据的类型。一些基本的有某种一致性的区域是事先给定的，然后用不同的方法加入周围的邻域。用边界松弛法分割：给定一个门限 Dmax，如果平均距离 $D(R)<D$max，则区域 R 是一致的（即是同一类）。如果 p 属于区域 Ra 且 p 与区域 Rb 相毗邻，并且将 p 从区域 Ra 移动到 Rb 的距离减去 $D(Ra)+D(Rb)$后还可以满足 $D(R)<D$max，那么我们就将 p 从区域 Ra 移至 Rb。用 2×2 的块，通过加入相邻的像素来增长区域：如果像素点的颜色值距离矩心的颜色值的差值小于 2Dmax，则加入该像素。除此之外，还可以用地理分水岭的算法来分割图像，该算法要求事先知道种子点的相关性；还可以根据一些统计数据来进行区域增长。或者用快速混合分割方法：由六边形邻域的均值代替该点的像素值（即平滑作用），这种分割方法是基于一种分等级的六边形结构组织，在最低层的时候，用局部区域增长法，这样可以获得小尺寸的相连区域；再上一层的时候，刚刚那层的每一个小尺寸相连区域被看作一个像素处理，同样再取六边形均值，再进行区域增长，这样一步一步进行下去。

区域分离与合并：前面讨论的区域生长过程是从一组种子点开始的，另一种可作为替换的方法是在开始时将图像分割为一系列任意不相交的区域，然后将它们进行聚合或拆分。还

有一种方法是先将图像分成彩色和非彩色区域，然后用直方图门限法，进行 8×8 大小块的合并，使用的色彩空间是 HSI。

3. 基于边缘的分割方法

边缘检测对彩色图像分割是一个重要的工具，其分割方法可以分为两大类：一个是局部边缘检测，另一个是全局边缘检测技术。边缘检测技术常用到 Sobel、高斯拉普拉斯算子。局部边缘检测技术只需要考虑像素点邻域的信息来决定一个边缘点，常用模糊 C-均值聚类算法。全局边缘检测技术必须考虑到全局最优化，一般来说，很多全局边缘检测算法是基于马尔可夫随机过程的不同应用。还有的算法是先用 Canny 算子检测出强度边缘，然后在提取出来的所有边缘里根据色调和饱和度门限逐步消除掉一些边缘。

4. 其他方法

除了上述三类彩色图像分割方法之外，还有一些彩色分割的综合方法，如分步分割方法：第一步借助取阈值方法进行粗略分割将图像转化为若干个区域，第二步利用模糊 C-均值法将第一步剩下的像素进一步分类，这种方法可看作是由粗到细进行的，先用取阈值方法是为了减少运用模糊 C-均值聚类所需的计算量。

测量空间聚类法是分割彩色图像常用的方法，彩色图像在各个空间均可看作由三个分量构成，所以分割彩色图像的一种方法是建立一个 3-D 直方图，它可用一个 3-D 数组表示。这个 3-D 数组中的每个元素代表图像中给定三个分量值的像素的个数。阈值分割的概念可以扩展为在 3-D 空间搜索像素的聚类，并根据聚类来分割图像。该方法的优点是：将图像由图像空间转换到测量空间的变换常是多对一的变换，这样变换后数据量减少易于计算；其次，尽管许多聚类方法本质上是递归或迭代的，大部分聚类方法可以产生比较光滑的区域边界且不易受噪声和局部边缘变化的影响。

马尔可夫随机场方法也可用于多分辨率彩色图像分割，这是一种收敛于最大后验概率的松弛方法。第一步的粗分割使用尺度空间滤波器以获得聚类的全局信息，第二步利用多尺度马尔可夫随机场细化分割。

第8章 图像复原

成像系统受各种因素的影响，导致了图像质量的降低，或者说是退化。由于获得图像的方法不同（光学、光电子或电子等），有多种退化形式，如传感器噪声、摄像机未聚焦、物体与摄像机之间的相对移动、随机大气湍流、光学系统的相差、成像光源和射线的散射等，都使成像的分辨率和对比度退化。这里读者要注意区分图像复原和图像增强两个概念的区别:

图像复原，试图利用退化过程的先验知识，去复原已被退化图像的本来面目。

图像增强，用适当方式改善图像质量，增强图像的视觉效果，以适应人眼的视觉和心理，不用考虑增强处理后的图像是否符合原有图像，是否失真。

图像退化的主要表现形式可分为图像模糊和图像受到噪声干扰。由于成像系统造成图像退化的典型现象是模糊，所以图像复原的一个基本任务就是去模糊。图像复原的基本思路:先建立退化的数学模型，然后根据该模型对退化图像进行拟合。图像复原模型可以用连续数学和离散数学处理，处理项的实现可在空间域卷积，或在频域相乘。

8.1 图像退化模型

如图 8-1 所示，退化过程可以被模型化为一个退化函数和一个加性噪声项，处理一幅输入图像 $f(x,y)$，产生一幅退化图像 $g(x,y)$。给定 $g(x,y)$和关于退化函数 H 的一些知识以及外加噪声项 $n(x,y)$，图像复原的目的是获得关于原始图像的近似估计 $\hat{f}(x,y)$。通常我们希望这一估计尽可能接近原始输入图像，并且 H 和 $n(x,y)$的信息知道得越多，所得到的估计 $\hat{f}(x,y)$ 就会越接近 $f(x,y)$。

$f(x,y) \rightarrow \boxed{h} \rightarrow \oplus \rightarrow g(x,y)$，其中 $n(x,y)$ 由下方输入 $\oplus$

图 8-1 图像退化模型

如果系统 H 是一个线性、位置不变性的过程，那么在空间域中给出的退化图像可由下式给出：

$$g(x,y)=h(x,y)*f(x,y)+n(x,y) \tag{8.1}$$

其中，$h(x,y)$是退化函数的空间描述；“ * ” 表示空间卷积。这是连续形式下的表达，在实际应用中，处理的都是数字图像，所以需对上式进行离散化如下：

$$g(x,y)=\sum_{p=0}^{M-1}\sum_{q=0}^{N-1}h(p,q)\cdot f(x-p,y-q)+n(x,y) \tag{8.2}$$

我们知道，空间域上的卷积等同于频域上的乘积，因此可以把退化模型式写成如下的频域表示：

$$G(u,v)=H(u,v)F(u,v)+N(u,v) \tag{8.3}$$

公式中的 $G(u,v)$、$H(u,v)$、$F(u,v)$和 $N(u,v)$分别是 $g(x,y)$、$h(x,y)$、$f(x,y)$和 $n(x,y)$的傅立叶变换。

数字图像的噪声主要来源于图像的获取（数字化过程）和传输过程。图像传感器的工作

情况受各种因素的影响，如图像获取中的环境条件和传感元器件自身的质量。例如，使用 CCD 摄像机获取图像，光照程度和传感器温度是生成图像中产生大量噪声的主要因素。图像在传输过程中主要由于所用的传输信道的干扰受到噪声污染。比如，通过无线网络传输的图像可能会因为光或其他大气因素的干扰被污染。典型的噪声有高斯噪声、瑞利噪声、伽马噪声、指数分布噪声、均匀分布噪声以及脉冲噪声（椒盐噪声）等。

8.1.1 线性、位置不变的退化

复原前，图 8-1 中的输入输出可以表示为：

$$g(x,y)=H\left[f(x,y)\right]+n(x,y) \tag{8.4}$$

现在，假设 $n(x,y)=0$，则 $g(x,y)=H\left[f(x,y)\right]$。如果

$$H\left[af_1(x,y)+bf_2(x,y)\right]=H\left[f_1(x,y)\right]+H\left[f_2(x,y)\right] \tag{8.5}$$

则系统 H 是一个线性系统。这里，a 和 b 是比例常数，$f_1(x,y)$ 和 $f_2(x,y)$ 是任意两幅输入图像。

对于任意 $f(x,y)$，α 和 β，如果

$$H\left[f(x-\alpha,y-\beta)\right]=g(x-\alpha,y-\beta) \tag{8.6}$$

则 $H\left[f(x,y)\right]=g(x,y)$ 称为位移不变系统。

由脉冲函数的定义我们知道，$f(x,y)$ 可描述为连续冲激函数：

$$f(x,y)=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}f(\alpha,\beta)\delta(x-\alpha,y-\beta)\mathrm{d}\alpha\mathrm{d}\beta \tag{8.7}$$

则有

$$g(x,y)=H\left[f(x,y)\right]=H\left[\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}f(\alpha,\beta)\delta(x-\alpha,y-\beta)\mathrm{d}\alpha\mathrm{d}\beta\right] \tag{8.8}$$

由于前面假设 H 是线性算子，可以将上式变为

$$g(x,y)=H\left[f(x,y)\right]=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}f(\alpha,\beta)H\left[\delta(x-\alpha,y-\beta)\right]\mathrm{d}\alpha\mathrm{d}\beta \tag{8.9}$$

则

$$h(x,y,\alpha,\beta)=H\left[\delta(x-\alpha,y-\beta)\right] \tag{8.10}$$

称为系统 H 的冲激响应。在光学中，冲激为一光点，所以 $h(x,y,\alpha,\beta)$ 一般称为点扩散函数（PSF）。

许多退化模型可近似表示为线性的位移不变过程。这一方法的优点在于应用了广泛使用的线性系统理论工具，对于解决图像复原问题很实用。非线性的与位置有关的技术，虽然更加普遍（通常会更精确），但是通常会对问题的求解带来很多困难，本书并不尝试解决这类复杂问题，有兴趣的读者可以参考相关文献。

8.1.2 图像几何畸变的退化

产生几何畸变的原因有很多种，例如，由于摄像机的扫描偏转系统有一定的非线性，会出现所谓桶形失真、枕形失真；或者由于地球表面呈球形，而卫星摄取的地球表面图像往往覆盖了较大面积，这样的平面图像会有较大的几何失真。几何畸变的复原或校正方法，需要针对考察具体的畸变原因，利用线性校正或双线性校正等方法，常以某一幅图像为基准，去

校正另一种摄取方式得到的图像的几何畸变，通常借用图像空间域变换操作进行处理，这部分内容在本书第三章做了一些介绍。另外，几何畸变的复原（校正）实现比较复杂，本书将不予过多阐述，感兴趣的读者可查阅图像处理与分析方面的书籍。

8.2 图像复原的方法

8.2.1 估计退化函数

在图像复原中，有 3 种主要的估计退化函数的方法：（1）观察法，（2）试验法，（3）数学建模法。在进行图像复原时，可以依据估计得到的退化函数进行图像去模糊操作。另外，也有一种图像复原方法很少或没有先验的退化模型函数做指导，这类方法称为盲目去卷积法。这种方法适用的问题更普遍，因为真正的退化函数很少能完全知晓，但处理效果往往并不能得到很满意的效果。

（1）图像观察估计法

假设提供了一幅退化图像，而没有退化函数 H 的知识，那么估计该函数的一个方法就是收集图像自身的信息。例如，如果图像是模糊的，可以观察包含简单结构的一小部分图像，如某一物体和背景的一部分。为了减少观察时的噪声影响，可以寻找强信号内容区。使用目标和背景的样品灰度级，可以构建一个不模糊的图像，该图像和看到的子图像有相同大小和特性。用 $g_s(x,y)$ 定义观察的子图像，用 $\hat{f}_s(x,y)$ 表示构建的子图像（现实中，它是原始图像在该区域的估计图像）。由于选择的是强信号区，信噪比非常高，可以近似忽略噪声效果，得局部退化模型函数为

$$H_s(u,v)=\frac{G_s(u,v)}{\hat{F}_s(u,v)} \tag{8.11}$$

假设这一函数具有位移不变性，则可以推出完全函数 $H(u,v)$。例如，假设 $H_s(u,v)$的径向曲线显示出巴特沃思低通滤波器的形状，我们就可以利用这一观察信息假设在整幅图像上构建函数 $H(u,v)$。

（2）试验估计法

如果可以使用与获取退化图像的设备相似的装置，理论上得到一个准确的退化估计是可能的。与退化图像类似的图像可以通过各种系统设置得到，退化这些图像使其尽可能接近希望复原的图像。利用相同的系统设置，由成像一个脉冲（小亮点）得到退化的冲激响应。因为线性的空间不变系统完全由它的冲激响应来描述。一个冲激可由一个亮点来模拟，并使它尽可能亮，以减少噪声的干扰。回顾一下，由于冲激的傅立叶变换是一个常量，则可得退化函数为：

$$H(u,v)=\frac{G(u,v)}{A} \tag{8.12}$$

这里，函数 $G(u,v)$与前面一样，是观察图像的傅立叶变换。A 是一个常量，表示冲激强度。

（3）模型估计法

退化模型通常是解决图像复原问题的关键，利用一个准确的退化模型，在没有强噪声干

扰的情况下，可以保证得到较好的复原效果。在某些情况下，模型要把引起退化的环境因素考虑在内。例如，Hufnagel 和 Stanley[1964]提出的退化模型是基于大气湍流的物理特性。这个模型有一个通用公式：

$$H(u,v)=\mathrm{e}^{-\mathrm{k}(u^2+v^2)^{5/6}} \tag{8.13}$$

这里，k 是常数，它与湍流的性质有关。除了指数为 5/6 次方之外，这个公式与高斯低通滤波器有相似的形式。

模型化的另一个主要方法是从基本原理开始推导一个数学模型。通过在某些细节上处理一个实例来解释这一过程，在该例中，图像获取时被图像与传感器之间的均匀线性运动模糊了。假设图像 $f(x,y)$进行平面运动，$x_0(t)$和 $y_0(t)$分别是在 x 和 y 方向上相应的随时间变化的运动参数。那么，在记录介质（如胶片或数字存储器）任意点的曝光总数是通过对时间间隔内瞬时曝光数的积分得到的，在该时间段，图像系统的快门是开着的。

假设快门的开启和关闭所用时间非常短，那么光学成像过程不会受到图像运动的干扰，非常完美。如果，设 T 为曝光时间，结果为：

$$g(x,y)=\int_0^T f\left[x-x_0(t),y-y_0(t)\right]\mathrm{d}t \tag{8.14}$$

$g(x,y)$为模糊的图像。

对上式进行傅立叶变换为：

$$\begin{aligned}G(u,v)&=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}g(x,y)\mathrm{e}^{-\mathrm{j}2\pi(ux+vy)}\mathrm{d}x\mathrm{d}y\\&=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\left[\int_0^T f\left[x-x_0(t),y-y_0(t)\right]\mathrm{d}t\right]\mathrm{e}^{-\mathrm{j}2\pi(ux+vy)}\mathrm{d}x\mathrm{d}y\end{aligned} \tag{8.15}$$

改变积分顺序，上式变为：

$$G(u,v)=\int_0^T\left[\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}f\left[x-x_0(t),y-y_0(t)\right]\mathrm{e}^{-\mathrm{j}2\pi(ux+vy)}\mathrm{d}x\mathrm{d}y\right]\mathrm{d}t \tag{8.16}$$

用外层括号内的积分项函数 $f\left[x-x_0(t),y-y_0(t)\right]$ 的傅立叶变换进行置换，得到如下表达式：

$$\begin{aligned}G(u,v)&=\int_0^T F(u,v)\mathrm{e}^{-\mathrm{j}2\pi[ux_0(t)+vy_0(t)]}\mathrm{d}t\\&=F(u,v)\int_0^T \mathrm{e}^{-\mathrm{j}2\pi[ux_0(t)+vy_0(t)]}\mathrm{d}t\end{aligned} \tag{8.17}$$

令

$$H(u,v)=\int_0^T \mathrm{e}^{-\mathrm{j}2\pi[ux_0(t)+vy_0(t)]}\mathrm{d}t \tag{8.18}$$

则可得表达示为

$$G(u,v)=H(u,v)F(u,v) \tag{8.19}$$

举例：若运动变量 $x_0(t)$和 $y_0(t)$已知，传递函数 $H(u,v)$可直接得到。假设当前图像只在 x 方向以给定的速度 $x_0(t)=at/T$ 做匀速直线运动。当 $t=T$ 时，图像由总距离 a 取代。令 $y_0(t)=0$，式（8.18）可变为：

$$\begin{aligned}H(u,v)&=\int_0^T \mathrm{e}^{-\mathrm{j}2\pi ux_0(t)}\mathrm{d}t\\&=\int_0^T \mathrm{e}^{-\mathrm{j}2\pi uat/T}\mathrm{d}t\end{aligned}$$

$$= \frac{T}{\pi ua}\sin(\pi ua)e^{-j\pi ua} \tag{8.20}$$

若允许 y 分量也变化，按 $y_0(t)=bt/T$ 运动，则退化函数变为：

$$H(u,v)=\frac{T}{\pi(ua+vb)}\sin[\pi(ua+vb)]e^{-j\pi(ua+vb)} \tag{8.21}$$

8.2.2 逆滤波

逆滤波是研究复原由退化函数 H 退化的图像的第一步。最简单的方法是直接逆滤波，即用退化函数 $H(u,v)$除退化图像的傅立叶变换 $G(u,v)$来计算原始图像的傅立叶变换估计 $\hat{F}(u,v)$：

$$\hat{F}(u,v)=\frac{G(u,v)}{H(u,v)}=F(u,v)+\frac{N(u,v)}{H(u,v)} \tag{8.22}$$

从该式可以看到，即使知道退化函数，也不能准确地复原被退化的图像，因为 $N(u,v)$是一个随机函数，它的傅立叶变换未知。另外，如果退化函数接近于零值，则 $N(u,v)/H(u,v)$成为主要成分，将造成 $\hat{F}(u,v)$ 的巨大波动。

8.2.3 最小均方误差滤波（维纳滤波）

前面讨论的逆滤波并没有清楚地说明怎样处理噪声，本节将讨论退化函数和噪声统计特征两个方面进行复原处理的方法。方法建立在认为图像和噪声是随机过程的基础上，而目标是找到一个污染图像 f 的估计值 $\hat{f}$，使它们之间的均方误差最小。误差度量由下式给出：

$$e^2=E\left[\left(f-\hat{f}\right)^2\right] \tag{8.23}$$

$E[\bullet]$是变量的期望值。这里假定：噪声和图像不相关，其中一个有零均值，估计的灰度级是退化图像灰度级的线性函数。在这些条件下，上式中误差函数的最小值在频域用下式计算：

$$\begin{aligned}\hat{F}(u,v)&=\left[\frac{H^*(u,v)S_f(u,v)}{S_f(u,v)|H(u,v)|^2+S_\eta(u,v)}\right]G(u,v)\\&=\left[\frac{H^*(u,v)}{|H(u,v)|^2+S_\eta(u,v)/S_f(u,v)}\right]G(u,v)\sqrt{b^2-4ac}\\&=\left[\frac{1}{H(u,v)}\frac{|H(u,v)|^2}{|H(u,v)|^2+S_\eta(u,v)/S_f(u,v)}\right]G(u,v)\end{aligned} \tag{8.24}$$

这里，我们应用了这样一个事实：一个复数量与它的共轭的乘积等于复数量幅度的平方。这个结果就是维纳滤波，是由 N.Wiener[1942]首次提出的。由括号里边的项组成的滤波器通常还叫做最小均方误差滤波器。注意上式的第一行，维纳滤波器没有逆滤波中退化函数为零的问题，除非对于相同的 u,v 值，$H(u,v)$和 $S_\eta(u,v)$ 都是 0。上式中的各项说明如下。

$H(u,v)$：退化函数。

$G(u,v)$：退化图像的变换。

$H^*(u,v)$：$H(u,v)$的复共轭。

$|H(u,v)|^2 = H(u,v)H^*(u,v)$

$S_\eta(u,v) = |N(u,v)|^2$：噪声的功率谱。

$S_f(u,v) = |F(u,v)|^2$：未退化图像的功率谱。

在空间域被复原的图像由频率域估计值 $\hat{F}(u,v)$ 的傅立叶反变换给出。注意，如果没有噪声，噪声功率谱消失，则维纳滤波退化为逆滤波。

当处理白噪声时，噪声功率谱 $|N(u,v)|^2$ 是一个常数，这大大简化了处理过程。然而，未退化图像的功率谱很少是已知的。当这些值未知或难于估计时，经常用下式近似：

$$\hat{F}(u,v) = \left[\frac{1}{H(u,v)} \frac{|H(u,v)|^2}{|H(u,v)|^2 + K}\right] G(u,v) \tag{8.25}$$

这里，K 表示一个特殊常数。

8.2.4 约束最小二乘滤波器

约束最小二乘滤波器与维纳滤波器同属约束滤波复原方法，而约束复原方法的通用表达式为：

$$\hat{F}(u,v) = \left[\frac{1}{H(u,v)} \frac{|H(u,v)|^2}{|H(u,v)|^2 + \frac{1}{\lambda}\frac{S_\eta(u,v)}{S_f(u,v)}}\right] G(u,v) \tag{8.26}$$

维纳滤波器对应的是 $\lambda = 1$，其所得到的估计值是使 $e^2 = E\left[\left(f - \hat{f}\right)^2\right]$ 取最小值时的最优估计；对于上式，当 $\lambda \neq 0$ 时，得到的估计称为变参量维纳滤波器，也称为约束最小二乘滤波器。

8.2.5 Lucy-Richardson 滤波复原

Lucy-Richardson 算法是目前应用最广泛的图像复原技术之一，采用迭代的方法。Lucy-Richardson算法能够按照泊松噪声统计标准求出与给定PSF卷积后最有可能成为输入模糊图像的图像。当 PSF 已知而图像噪声信息未知时，也可以采用该算法进行复原操作。从成像方程和泊松统计可以有如下推导：

$$I(i) = \sum_j P(i|j)O(j) \tag{8.27}$$

式中，O 是原始图像，$P(i|j)$ 是 PSF，I 是无噪声模糊图像。在已知 $I(i)$时，在每个图像像素点的估计 $D(i)$的联合似然函数为

$$\ln\prod = \sum_i \left[D(i)\ln I(i) - I(i) - \ln D(i)\right] \tag{8.28}$$

当下式满足时，其最大似然函数的解存在。

$$\frac{\partial \ln \prod}{\partial O(j)} = \sum_i \left[\frac{D(i)}{I(i)} - 1 \right] P(i|j) = 0 \tag{8.29}$$

则可得 Lucy-Richardson 迭代式，即

$$O_{new}(j) = O(j) \frac{\sum_i P(i|j) D(i)/I(i)}{\sum_i P(i|j)} \tag{8.30}$$

可以看出每次迭代时，都可以提高解的似然性。随着迭代次数的增加，最终将会收敛在具有最大似然性的解处。

8.3 用于图像复原的 MATLAB 工具箱函数

8.3.1 图像复原函数

MATLAB7.0 的图像处理工具箱提供了 4 个图像复原函数，按照其复杂程度列举如下。

- deconvwnr 函数：使用维纳滤波器；
- deconvreg 函数：使用约束最小二乘滤波器；
- deconvlucy 函数：使用 Lucy-Richardson 滤波器；
- deconvblind 函数：使用盲卷积算法。

以上所有复原函数大多需要以 PSF 和模糊图像作为主要输入参数，在图像复原过程中，都应该向其提供噪声的有关信息。

（1）deconvwnr

该函数可以实现最小均方误差滤波。其调用格式为：

J = deconvwnr(I,PSF)

J = deconvwnr(I,PSF,NSR)

J = deconvwnr(I,PSF,NCORR,ICORR)

其中，I 表示输入图像，PSF 表示点扩散函数，NSR 表示输入图像的信噪比的倒数，即噪声功率比信号功率，默认值为 0，NCORR 和 ICORR 是可选参数，分别表示噪声的自相关函数和输入图像的自相关函数。输出 J 表示被复原的图像。

（2）deconvreg

在使用该函数时，可以对输出图像采用某些约束，比如默认情况下为光滑性约束。其调用格式为：

J = deconvreg(I,PSF)

J = deconvreg(I,PSF,NOISEPOWER)

J = deconvreg(I,PSF,NOISEPOWER,LRANGE)

J = deconvreg(I,PSF,NOISEPOWER,LRANGE,REGOP)

[J, LAGRA] = deconvreg(I,PSF,···)

其中，I 表示输入图像，PSF 表示点扩散函数，NOISEPOWER 表示图像 I 中的加性噪声

功率，默认值是 0，LRANGE 是一个矢量，表示执行最优解搜索的范围，默认值是[e^{-9}, e^{9}]，REGOP 是约束反卷积的规则化算子，默认值为拉普拉斯算子。LAGRA 表示返回得到拉格朗日乘算子，J 表示复原图像。

（3）deconvlucy

该函数可以实现一个加速收敛的 Lucy-Richardson 算法，该函数将使用最优化技术和泊松统计完成多次重复过程，但该函数不需模糊图像的噪声信息。其调用格式如下：

```
J = deconvlucy(I,PSF)
J = deconvlucy(I,PSF,NUMIT)
J = deconvlucy(I,PSF,NUMIT,DAMPAR)
J = deconvlucy(I,PSF,NUMIT,DAMPAR,WEIGHT)
J = deconvlucy(I,PSF,NUMIT,DAMPAR,WEIGHT,READOUT)
J = deconvlucy(I,PSF,NUMIT,DAMPAR,WEIGHT,READOUT,SUBSMPL)
```

其中，I 表示输入图像，PSF 表示点扩散函数。其他参数都是可选参数：NUMIT 表示反卷积函数的迭代次数，默认值为 10；DAMPAR 指定了收敛过程中结果图像与原始图像背离程度的阈值，对于那些超过阈值的数据，将不再允许进行反复计算，默认值为 0（无偏差），该参数的合理设定可以有效控制低信噪比条件下输出图像中出现的斑点噪声；WEIGHT 表示像素加权值，默认值为与输入图像大小相同的元素全为 1 的矩阵，通过指定某些对应像素的加权值为 0 可以忽略图像中可能包含的坏像素点；READOUT 表示相机成像噪声和背景噪声的总和，默认值为 0；SUBSMPL 指定采样不足的比例，默认值为 1。

通常把输入图像 I 作为函数参数，则输出 J 为复原图像矩阵；当把元数组 cell array 作为输入参数时，函数的输出 J 也是一个元数组，包含 4 个元素：output(1)，原始输入图像；output(2)，最后一次反复产生的图像；output(3)，倒数第二次产生的图像；output(4)，deconvlucy 函数用来获知重新起始点的内部信息。输出的单元数组可以作为输入参数传递给 deconvlucy 函数，从而重新开始反复计算过程。

（4）deconvblind

该函数实现盲卷积算法，执行过程中可以不需要有关 PSF 的知识，调用时将 PSF 的一个初始化估计作为输入参数即可，同时，该函数还给复原后的图像返回一个重建的 PSF，实现过程中使用了与 deconvlucy 函数相同的收敛方式。其调用格式如下：

```
[J,PSF] = deconvblind(I,INITPSF)
[J,PSF] = deconvblind(I,INITPSF,NUMIT)
[J,PSF] = deconvblind(I,INITPSF,NUMIT,DAMPAR)
[J,PSF] = deconvblind(I,INITPSF,NUMIT,DAMPAR,WEIGHT)
[J,PSF] = deconvblind(I,INITPSF,NUMIT,DAMPAR,WEIGHT,READOUT)
[J,PSF] = deconvblind(…,FUN,P1,P2,…,PN)
```

其中，I 表示输入图像，INITPSF 表示 PSF 的估计值，NUMIT 表示算法的迭代次数，默认值为 10。与函数 deconvlucy 的对应参数类似，在函数 deconvblind 中，DAMPAR 表示偏移阈值；WEIGHT 是一个与图像 I 大小相同的数组，其元素值在 0 和 1 之间，通过指定某些位置的值为 0 可以用来屏蔽坏像素；READOUT 表示噪声。J 表示复原图像，PSF 与 INITPSF 大小相同，表示重建的点扩散函数。FUN 表示对 PSF 的加性约束函数，它接受 PSF 和 P1,P2,…,PN 为参数，返回值与 PSF 大小相同。

除了以上 4 个复原函数以外，还可以使用 MATLAB 自定义复原函数。

8.3.2 图像模糊函数

在以下 MATLAB 的图像复原介绍中，为了能比较复原的效果和性能，一般给出原始图像和模糊后的图像以及利用工具箱的复原函数处理后的图像进行比较。为了创建模糊化的图像，通常使用 MATLAB 的图像工具箱函数 fspecial 创建一个确定类型的 PSF，然后使用这个 PSF 对原图像进行卷积，从而得到模糊化的图像。

以下程序代码分别产生运动模糊、圆盘状模糊以及钝化模糊的 PSF，然后将 PSF 与原图像卷积，得到不同的模糊图像，如图 8-2 所示。

```
I = imread('cameraman.tif');
subplot(2,2,1);
imshow(I); title('Original Image');
H = fspecial('motion',20,45);
MotionBlur = imfilter(I,H,'replicate');
subplot(2,2,2);
imshow(MotionBlur);title('Motion   Blurred
Image');
H = fspecial('disk',10);
blurred = imfilter(I,H,'replicate');
subplot(2,2,3);
imshow(blurred); title('Blurred Image');
H = fspecial('unsharp');
sharpened = imfilter(I,H,'replicate');
subplot(2,2,4);
imshow(sharpened); title('Sharpened Image');
```


Image Courtesy of MIT
Original Image


Motion Blurred Image


Blurred Image


Sharpened Image

图 8-2 图像模糊化

由前面的分析可知，如果图像中不存在噪声，则其模糊状况完全由 PSF 决定，此时，去模糊的主要任务是使用精确描述失真的 PSF 对模糊图像进行解卷积操作。

在实际应用中，需要复原的图像都是含有噪声的。在 MATLAB7.0 中，可以使用两种方法模拟图像噪声：

（1）使用 imnoise 函数，直接对图像添加固定类型的噪声。

（2）创建自定义的噪声，然后使用 MATLAB 图像代数运算函数 imadd 将其添加到图像中去。

下面的程序代码分别对原始图像中添加椒盐噪声、高斯噪声、泊松噪声以及斑点噪声，4 种添加噪声的效果图如图 8-3 所示。

```
I = imread('LENA256.bmp');
J1 = imnoise(I,'salt & pepper',0.02);
subplot(2,2,1);imshow(J1);title('salt & pepper noise');
J2 = imnoise(I,'gaussian',0.02);
subplot(2,2,2);imshow(J2);title('gaussian noise');
```

```
J3 = imnoise(I,'poisson');
subplot(2,2,3);imshow(J3);title('poisson noise');
J4 = imnoise(I,'speckle');
subplot(2,2,4);imshow(J4);title('speckle noise');
```

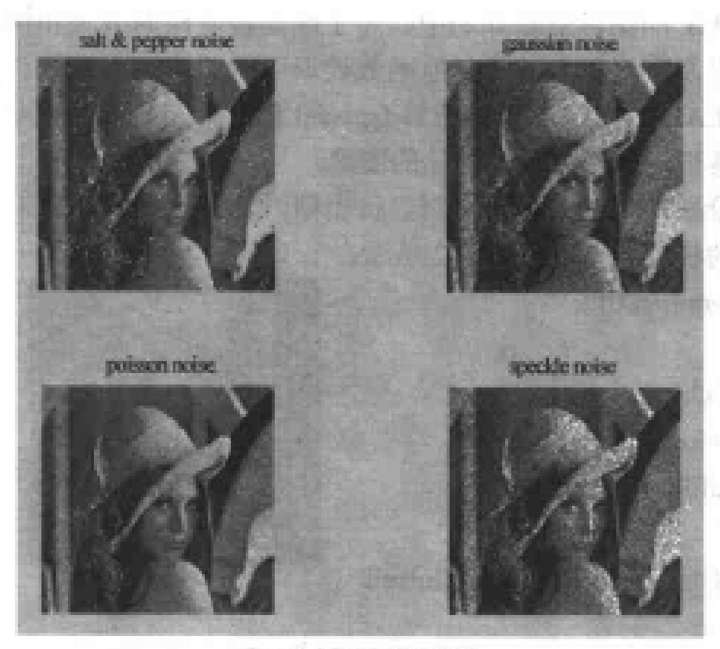


图 8-3 图像加噪声结果

8.4 图像复原应用

8.4.1 生成模糊化实验图像

为了实验各种滤波复原函数的效果，首先读取一幅原始图像并模糊化和加噪，如图 8-4 所示。实现方法如下：

（1）读取原始图像

```
I = imread('LENA256.bmp');
imshow(I);title('Original Image');
```



图 8-4 原始图像

（2）图像模糊化

分别进行“motion”（运动）模糊处理和“gaussian”模糊处理。所得结果如图 8-5 所示。

```
LEN = 31;
THETA = 11;
PSF1 = fspecial('motion',LEN,THETA);
PSF2 = fspecial('gaussian',10,5);
Blurred1 = imfilter(I,PSF1,'circular','conv');
Blurred2 = imfilter(I,PSF2,'conv');
```

```
subplot(1,2,1);imshow(Blurred1);title('Blurred1--"motion"');
subplot(1,2,2);imshow(Blurred2);title('Blurred2--"gaussian"');
```

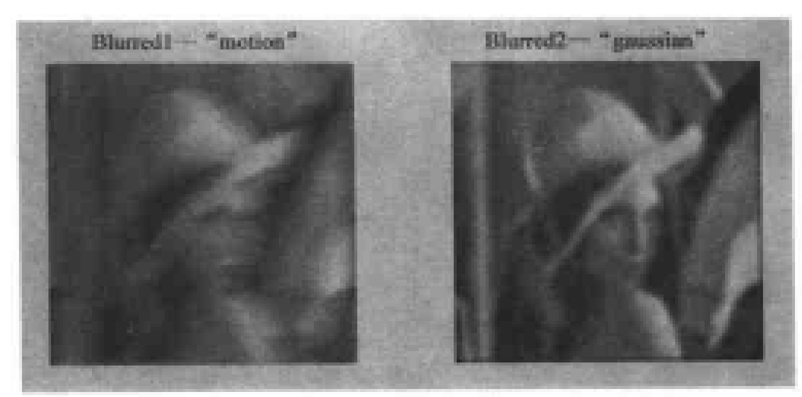
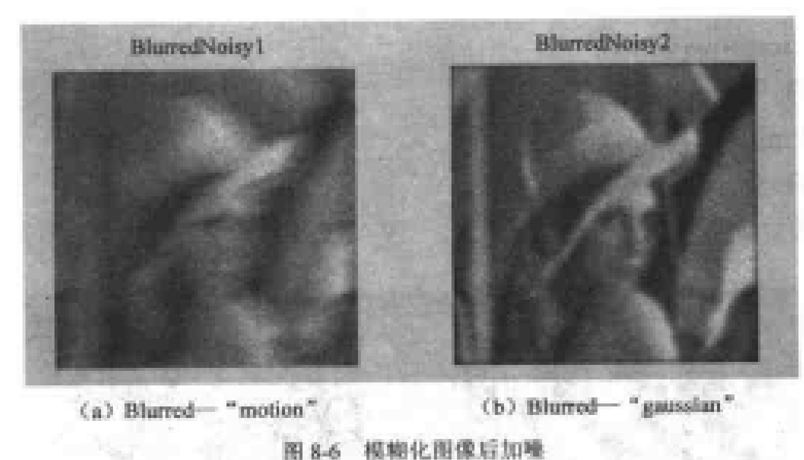


图 8-5 模糊化图像

（3）模糊化图像加噪

对上一步模糊化的图像增加加性“gaussian”噪声，所得结果如图 8-6 所示。

```
V = .002;
BlurredNoisy1 = imnoise(Blurred1,'gaussian',0,V);
BlurredNoisy2 = imnoise(Blurred2,'gaussian',0,V);
figure;
subplot(1,2,1);imshow(BlurredNoisy1);title('BlurredNoisy1');
subplot(1,2,2);imshow(BlurredNoisy2);title('BlurredNoisy2');
```



（a）Blurred— "motion"　　（b）Blurred— "gaussian"

图 8-6 模糊化图像后加噪

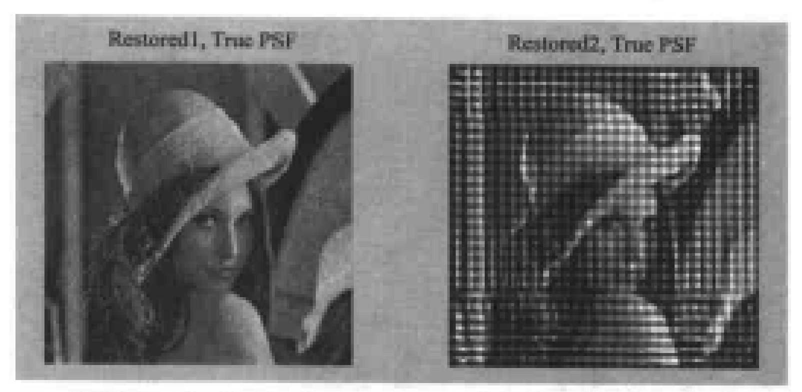
8.4.2 维纳滤波复原

对于模糊后的图像，用真实的 PSF 函数采用维纳滤波方法复原图像，程序代码如下：

```
wnr1 = deconvwnr(Blurred1,PSF1);
wnr2 = deconvwnr(Blurred2,PSF2);
subplot(1,2,1);imshow(wnr1);
title('Restored1, True PSF');
subplot(1,2,2);imshow(wnr2);
```

```
title('Restored2, True PSF');
```

图像复原效果如图 8-7 所示。



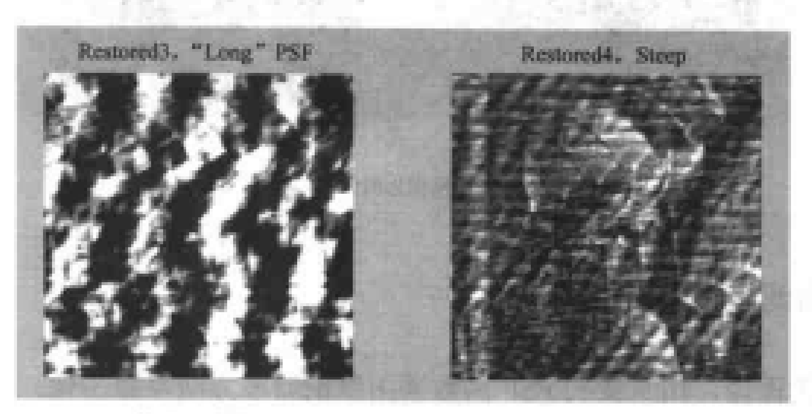
（a）Blurred—“motion”　　（b）Blurred—“gaussian”

图 8-7　采用真实 PSF 函数的维纳滤波复原（无噪声）

从复原的图像来看，对于运动引起的模糊图像的复原效果要好于因 gaussian 模糊的图像复原的效果。但这是假定预先知道引起模糊的精确点扩散函数，实际图像处理的任务中，大多数无法准确得知点扩散函数，一般采用估计的 PSF 来复原图像。下面的例子是针对因运动引起的图像模糊，利用估计的点扩散函数，相对于真实的点扩散函数，分别采用了过大的模糊距离参数和过大的模糊运动方向角度参数，实现代码如下：

```
wnr3 = deconvwnr(Blurred1,fspecial('motion',2*LEN,THETA));
wnr4 = deconvwnr(Blurred1,fspecial('motion',LEN,2*THETA));
figure;
subplot(1,2,1);imshow(wnr3);
title('Restored3, "Long" PSF');
subplot(1,2,2);imshow(wnr4);
title('Restored4, Steep');
```

图像复原的结果如图 8-8 所示。



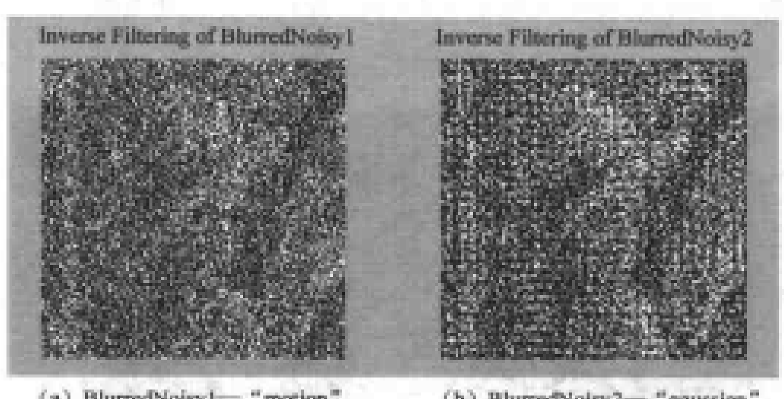
（a）过大的距离模糊参数　　（b）过大的模糊运动方向角度参数

图 8-8　利用估计的 PSF 复原模糊图像（无噪声）——（Blurred—“motion”）

如果模糊图像中同时含有加性噪声，如图 8-6 所示的图像，则需要在图像复原过程中考虑噪声的影响，即需要估计信噪比。下面我们来对比一下复原过程中是否考虑噪声影响的结果。实现代码如下：

```
%不考虑噪声的影响，采用逆滤波方法复原图像
wnr5 = deconvwnr(BlurredNoisy1,PSF1);
wnr6 = deconvwnr(BlurredNoisy2,PSF2);
figure;
subplot(1,2,1);imshow(wnr5);
title('Inverse Filtering of BlurredNoisy1');
subplot(1,2,2);imshow(wnr6);
title('Inverse Filtering of BlurredNoisy2');
```

复原图像结果如图 8-9 所示。



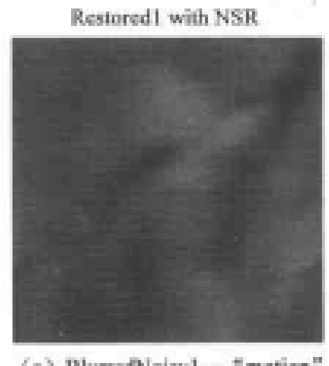
（a）BlurredNoisy1— "motion"　　（b）BlurredNoisy2— "gaussian"

图 8-9　不考虑噪声影响的复原结果

```
%估计信噪比，应用于图像复原过程中
NSR = sum((V*prod(size(I))).^2) / sum(im2double(I(:)).^2);%信噪比的倒数
wnr7 = deconvwnr(BlurredNoisy1,PSF1,NSR);
wnr8 = deconvwnr(BlurredNoisy2,PSF2,NSR);
figure;
subplot(1,2,1);imshow(wnr7);
title('Restored1 with NSR');
subplot(1,2,2);imshow(wnr8);
title('Restored2 with NSR');
```

将估计得到的信噪比的倒数作为参数得到的复原图像如图 8-10 所示。

Restored1 with NSR



Restored2 with NSR

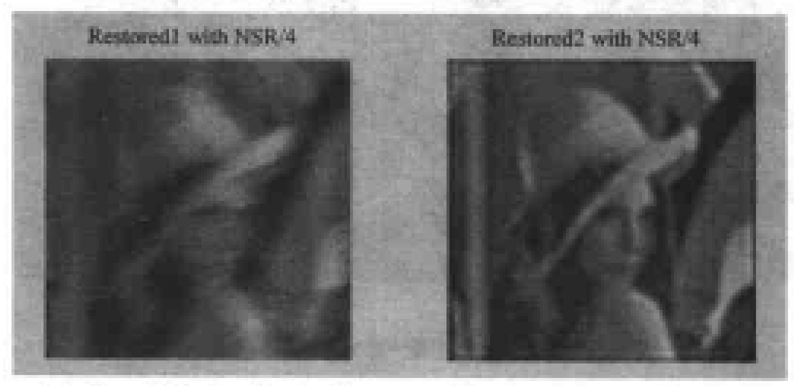


（a）BlurredNoisy1— "motion"　　（b）BlurredNoisy2— "gaussia"

图 8-10　利用估计的信噪比的倒数作参数的复原结果

```
%采用过小的噪信比（即过大的信噪比）作为参数
wnr9 = deconvwnr(BlurredNoisy1,PSF1,NSR/100);
wnr10 = deconvwnr(BlurredNoisy2,PSF2,NSR/100);
figure;
subplot(1,2,1);imshow(wnr9);
title('Restored1 with NSR/100');
subplot(1,2,2);imshow(wnr10);
title('Restored2 with NSR/100');
```

利用过小的噪信比（即过大的信噪比）作参数进行图像复原，结果如图 8-11 所示。由图可以看到，这个结果比利用估计过大噪信比的方法得到的结果更加明亮，但对噪声的抑制能力却明显不如后者。



（a）BlurredNoisy1— "motion"　　（b）BlurredNoisy2— "gaussian"

图 8-11　利用过小的噪信比（即过大的信噪比）作参数的复原图像

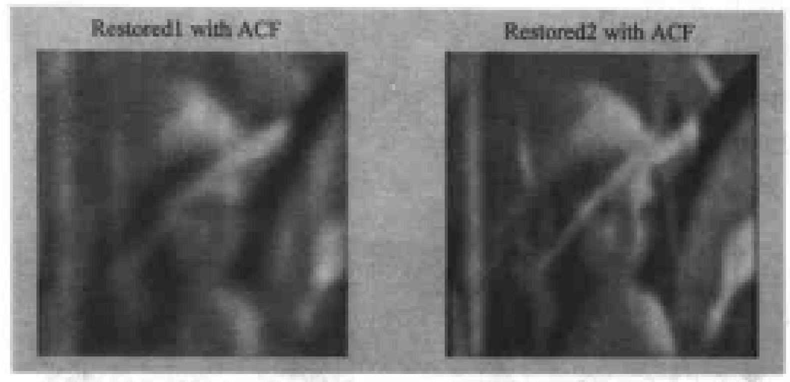
除了利用噪信比作参数进行图像复原，MATLAB 还提供了利用图像的自相关信息来提高图像复原质量的方法，这需要提供信号的自相关函数 ICORR 和噪声的自相关函数 NCORR。

```
NP=(V*prod(size(I))).^2;
NPOW = sum(NP(:))/prod(size(I)); % 噪声功率
NCORR = fftshift(real(ifftn(NP))); % 噪声自相关函数（ACF）
IP = abs(fftn(im2double(I))).^2;
IPOW = sum(IP(:))/prod(size(I)); % 原始图像的功率
ICORR = fftshift(real(ifftn(IP))); % 图像自相关函数（ACF）
wnr11 = deconvwnr(BlurredNoisy1,PSF1,NCORR,ICORR);
wnr12 = deconvwnr(BlurredNoisy2,PSF2,NCORR,ICORR);
figure;
subplot(1,2,1);imshow(wnr11);
title('Restored1 with ACF');
subplot(1,2,2);imshow(wnr12);
title('Restored2 with ACF');
```

复原结果如图 8-12 所示。

对图像进行复原操作时，知道的关于图像的统计信息越丰富，得到的复原结果就越好，反之则更差。如图 8-13 表示仅已知噪声的功率 NPOW 和真实图像的 1-D 自相关函数 ICORR

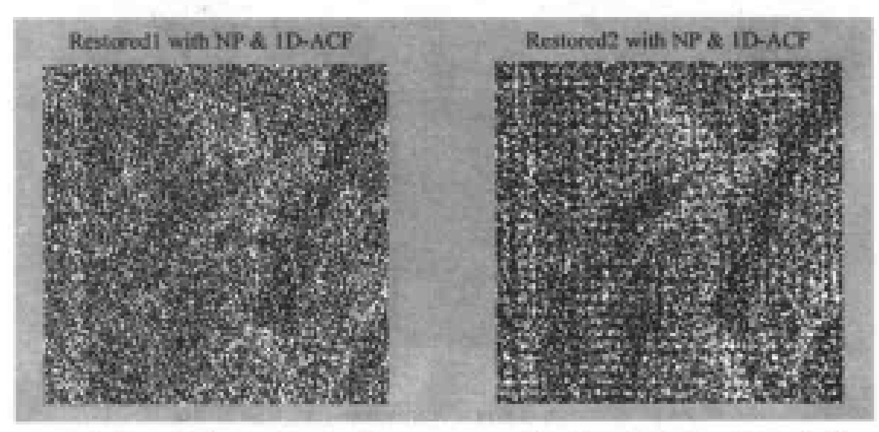
得到的复原图像。



（a）BlurredNoisy1— "motion"　　（b）BlurredNoisy2— "gaussian"

图 8-12　利用自相关信息得到的复原图像

```
ICORR1 = ICORR(:,ceil(size(I,1)/2));
wnr13 = deconvwnr(BlurredNoisy1,PSF1,NPOW,ICORR1);
wnr14 = deconvwnr(BlurredNoisy2,PSF2,NPOW,ICORR1);
figure;
subplot(1,2,1);imshow(wnr13);
title('Restored1 with NP & 1D-ACF');
subplot(1,2,2);imshow(wnr14);
title('Restored2 with NP & 1D-ACF');
```



（a）BlurredNoisy1— "motion"　　（b）BlurredNoisy2— "gaussian"

图 8-13　已知有限统计信息的复原图像

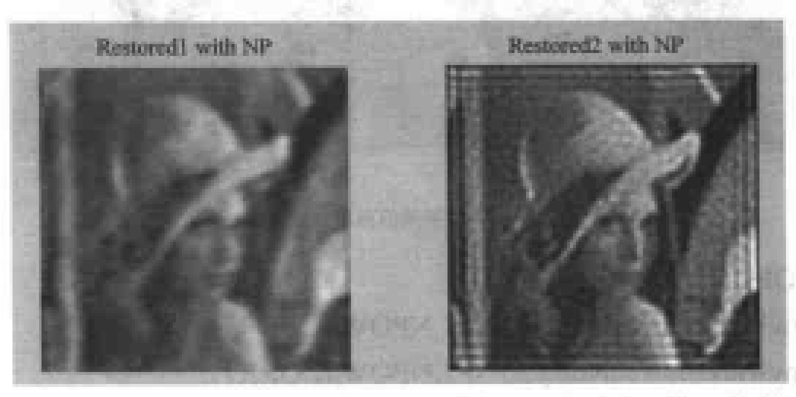
8.4.3　约束最小二乘滤波复原

以 8.4.1 节开始生成的模糊图像作为输入，利用约束最小二乘滤波复原图像的方法如下：

```
%用真实的 PSF 函数和噪声强度作为参数进行图像复原
NP = V*prod(size(I)); % noise power
reg1 = deconvreg(BlurredNoisy1,PSF1,NP);
reg2 = deconvreg(BlurredNoisy2,PSF2,NP);
```

```
figure;
subplot(1,2,1);imshow(reg1);
title('Restored1 with NP');
subplot(1,2,2);imshow(reg2);
title('Restored2 with NP');
```

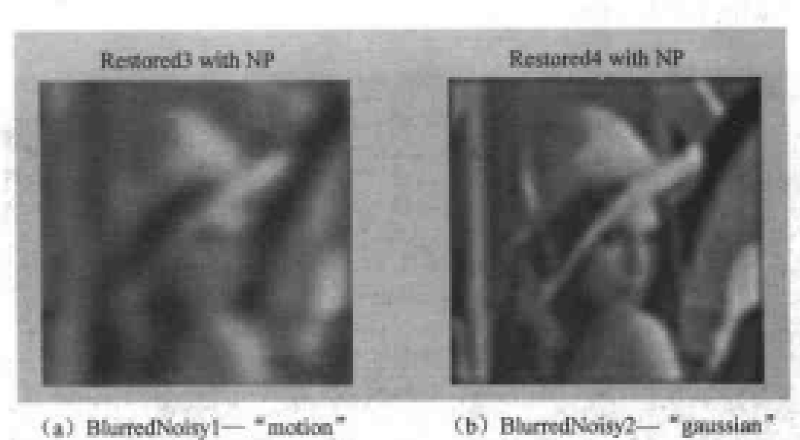
图像复原的结果如图 8-14 所示。



（a）BlurredNoisy1—“motion”　　（b）BlurredNoisy2—“gaussian”

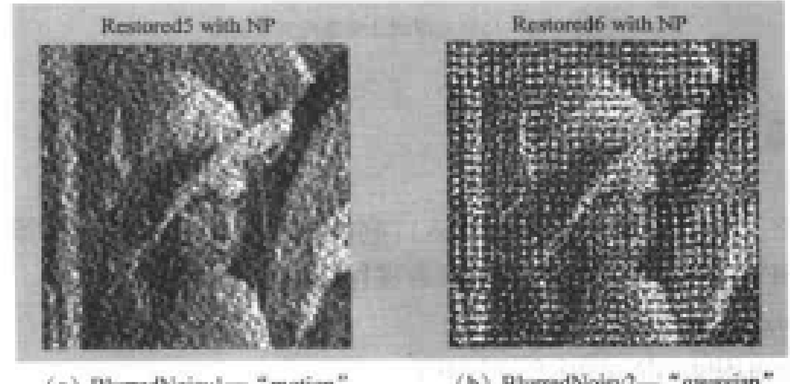
图 8-14　用真实的 PSF 函数和噪声强度作为参数的约束最小二乘滤波复原

当作为参数的图像噪声强度出现偏差时，图像复原的效果将会受到影响，结果如图 8-15（噪声强度参数过大，NP1=NP*1.3）和图 8-16（噪声强度参数过小，NP2= NP/1.3）所示。



（a）BlurredNoisy1—“motion”　　（b）BlurredNoisy2—“gaussian”

图 8-15　噪声强度参数过大（NP1=NP*1.3）的复原图像



（a）BlurredNoisy1—“motion”　　（b）BlurredNoisy2—“gaussian”

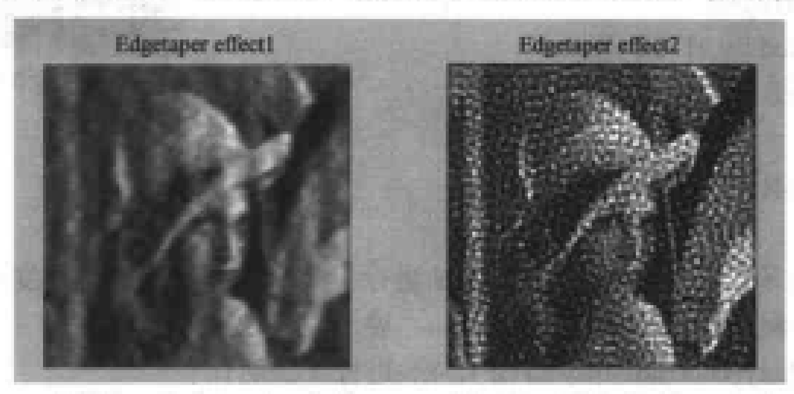
图 8-16　噪声强度参数过小（NP2=NP/1.3）的复原图像

通过图 8-15 和图 8-16 可以发现，过大的估计噪声强度会导致复原效果大大下降，而过小的估计噪声强度导致了噪声的放大效果，并且产生了振铃现象。

针对噪声估计过小引起的噪声放大效应和振铃现象，可以先通过调用 edgetaper 函数，对图像边缘信息进行提取，然后作为 deconvreg 函数的参数进行图像复原，在这种情况下，复原的效果将对噪声强度不敏感。代码实现如下：

```
Edged1 = edgetaper(BlurredNoisy1,PSF1);
Edged2 = edgetaper(BlurredNoisy2,PSF2);
reg7 = deconvreg(Edged1,PSF1,NP/1.3);
reg8 = deconvreg(Edged2,PSF2,NP/1.3);
figure;
subplot(1,2,1);imshow(reg7);
title('Edgetaper effect1');
subplot(1,2,2);imshow(reg8);
title('Edgetaper effect2');
```

处理结果如图 8-17 所示，可以看出，相比图 8-16 的处理结果，振铃现象得到明显抑制。



（a）BlurredNoisy1— "motion"　　（b）BlurredNoisy2— "gaussian"

图 8-17　振铃抑制复原图像

对于一个模糊加噪图像的复原操作，如果假设其最优解存在，并且已知相应的拉格朗日算子 LARGA，那么可以忽略关于噪声强度的参数传递，如 NP 等。下面的处理过程将说明复原效果对 LARGA 的依赖性。这里仅以一幅图像（Blurred by motion）说明。实现代码如下：

```
[reg1 LAGRA] = deconvreg(BlurredNoisy1,PSF1,NP);
reg9 = deconvreg(Edged1,PSF1,[],LAGRA);
reg10 = deconvreg(Edged1,PSF1,[],LAGRA*100);
reg11 = deconvreg(Edged1,PSF1,[],LAGRA/100);
figure;
subplot(1,3,1);imshow(reg9);
title('true LAGRA');
subplot(1,3,2);imshow(reg10);
title('large LAGRA');
subplot(1,3,3);imshow(reg11);
title('small LAGRA');
```

处理结果如图 8-18 所示。



图 8-18 不同拉格朗日算子复原的结果

分析上面 3 幅图像，可以发现过大的 LARGA 使得复原的图像过于平滑，而过小的 LARGA 使得复原的图像失去了利用的价值，噪声得不到抑制。

除了拉格朗日乘算子，也可以使用一维拉普拉斯算子来寻找最优复原结果。实现代码如下：

```
REGOP = [1 -2 1];
reg8 = deconvreg(BlurredNoisy,PSF,[],LAGRA,REGOP);
figure;imshow(reg8);
title('Constrained by 1D Laplacian');
```

处理结果如图 8-19 所示。



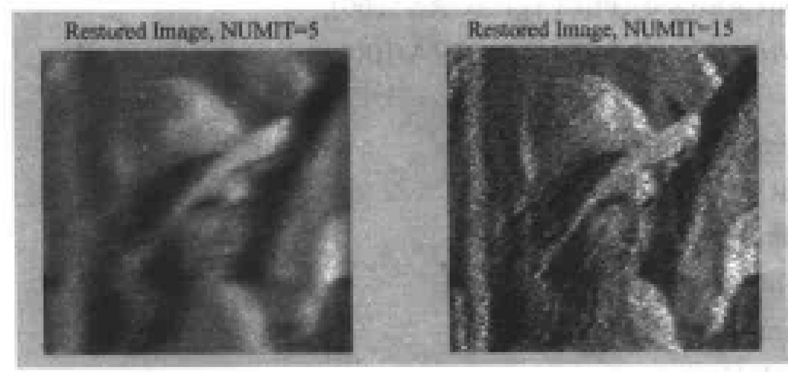
图 8-19 拉普拉斯算子复原图像

8.4.4 Lucy-Richardson 滤波器

该复原方法是采用 Lucy-Richardson 算法进行指定次数的迭代而得到复原图像，实现方法如下（这里仅以一幅图像，Blurred by motion，展示复原效果）：

```
luc1 = deconvlucy(BlurredNoisy1,PSF1,5);%进行 5 次迭代
luc2 = deconvlucy(BlurredNoisy1,PSF1,15);%进行 15 次迭代
figure;
subplot(1,2,1);imshow(luc1);
title('Restored Image, NUMIT = 5');
subplot(1,2,2);imshow(luc2);
title('Restored Image, NUMIT = 15');
```

处理结果如图 8-20 所示。



（a）5 次迭代结果　　（b）15 次迭代结果

图 8-20 Lucy-Richardson 滤波复原结果

在前面介绍的 deconvlucy 函数的参数中，我们了解到当以元数组（cell array）作为函数参数时，得到的输出也是元数组，同时，该输出结果可以再作为函数的输入参数，得到更进一步的输出结果。由此，使得我们可以通过这种方式实现对该滤波复原方法迭代过程的跟踪，显示中间计算结果和各类参数取值。假定，我们要实现 15 次迭代求解复原图像，而我们可以通过下面的方法实现对第 5 次迭代结果的查看，实现代码如下：

```
%恢复迭代过程，把前 5 次迭代结果作为输入
%默认迭代次数为 10 次
luc2_cell = deconvlucy(luc1_cell,PSF);
%元数组的第 2 个分量 luc2_cell{2}表示最终迭代结果图像，
%但其存储类型是 double，需要转换为 uint8 类型
luc2 = im2uint8(luc2_cell{2});
figure;imshow(luc2);title('Restored Image, NUMIT = 15');
```

得到的结果与图 8-20（b）中直接执行 15 次迭代的结果是一致的。

如图 8-20（b）所示，经过 15 次迭代的结果比迭代 5 次的结果更加锐化，然而图像中增加了斑点，这些斑点的出现并不是对应了真实的图像结构，而是因为真实图像与噪声数据太接近的结果。为了控制噪声的放大，可以通过指定参数 DAMPAR 来达到衰减噪声的目的。在应用时，DAMPAR 必须与输入图像的类型相同。下面的例子展示了利用参数 DAMPAR 的结果。代码如下：

```
DAMPAR = im2uint8(3*sqrt(V));%指定 DAMPAR 为噪声标准差的 3 倍
luc3 = deconvlucy(BlurredNoisy,PSF,15,DAMPAR);
figure;imshow(luc3);
title('Restored Image with Damping, NUMIT = 15');
```

处理结果如图 8-21 所示。

至于函数 deconvlucy 的其他参数的使用，读者可以进一步参考 MATLAB7.0 帮助文档。



图 8-21　使用参数 DAMPAR 的复原图像

8.4.5　盲卷积滤波复原

在本小节中，我们引入一幅新图像——“cameraman.tif”，下面通过几段程序代码实例来说明 deconvblind 函数的使用方法。

1. 图像模糊化

首先读取原始图像，如图 8-22（a）所示，生成如图 8-22（b）所示的点扩散函数 PSF，对图像进行模糊化，模糊结果如图 8-22（c）所示。

```
I = imread('cameraman.tif');
figure;imshow(I);title('Original Image');
PSF = fspecial('motion',13,45);
figure; imshow(PSF,[],'notruesize');
```

```
Blurred = imfilter(I,PSF,'circ','conv');
figure; imshow(Blurred); title('Blurred Image');
```





（a）原始图像　（b）真实 PSF　（c）模糊化图像

图 8-22　用于盲卷积滤波的图像模糊化

2. 图像复原

在调用 deconvblind 函数进行图像复原时，INITPSF 的大小是一个非常重要的指标。在实际应用中，通过分析，都是使用不同大小的 PSF 对图像进行重建，从中选择一个最合适的 PSF 值。以下程序段以真实大小的 INITPSF 复原图像，得到初步复原结果，如图 8-23 所示，同时初步重建 PSF，如图 8-24 所示。

```
INITPSF = ones(size(PSF));%获取函数的特征
[J P]= deconvblind(Blurred,INITPSF,30);%盲卷积，保留使用的 PSF
figure; imshow(J);
figure; imshow(P,[],'notruesize');
```



图 8-23　初步复原的图像

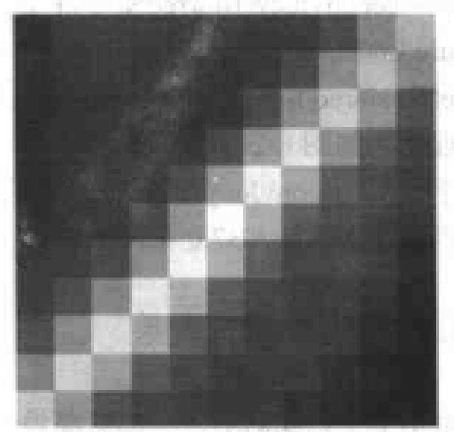


图 8-24　初步重建使用的 PSF

由图 8-23 可以看出，复原图像中存在一些“光环”，这是由图像灰度变换较大的部分或图像边界产生的。下面介绍如何使用 WEIGHT 参数来消除环的存在，以提高图像的复原质量。

首先调用 edge 函数找出图像中灰度变化较大的部分。根据先验知识，选择灰度变化阈值为 0.28。同时对图像进行膨胀作用以扩充图像的处理区域。灰度变换较大的像素和图像的边界像素都将被设置为数值 0。最终得到如图 8-25 所示的权值矩阵。

```
WEIGHT = edge(I,'sobel',.28);%sobel 算子提取边缘
se1 = strel('disk',1);
se2 = strel('line',13,45);
```

WEIGHT = ~imdilate(WEIGHT,[se1 se2]);%膨胀操作，边界像素设为零

WEIGHT = padarray(WEIGHT(2:end-1,2:end-1),[2 2]);

figure; imshow(WEIGHT);



图 8-25　权值矩阵

然后使用以上定义的 WEIGHT 数组对图像进行重建，得到如图 8-26 所示的复原结果。由图可以看出，复原后的图像消除了“环”的存在，但是复原结果仍然有一定的失真，图 8-27 表示复原图像使用的 PSF。

```
P1 = P;%保存数据
P1(find(P1 < 0.01))=0;%修改 PSF 函数
%利用上面得到的 WEIGHT 进行盲卷积
[J2 P2] = deconvblind(Blurred,P1,50,[],WEIGHT);
figure; imshow(J2);
figure; imshow(P2,[],'notruesize');
```



图 8-26　最终得到的复原图像

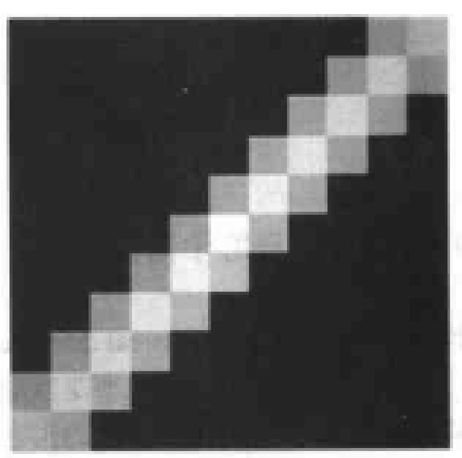


图 8-27　复原图像使用的 PSF

第9章 数学形态学图像处理

近年来，形态学图像处理已发展成为图像处理的一个重要研究领域，关于形态学理论及应用的文章大量出现在各种研究期刊和会议中，许多应用系统也用到了形态学理论。目前形态学的应用几乎覆盖了图像处理的所有领域，包括医学图像处理、文字识别、图像编码压缩、材料科学、视觉检测以及计算机视觉等。一些图像分析系统还将数学形态运算作为系统的基本运算，并由此出发考虑系统的体系结构。由此可以看出，形态学方法已经成为从事图像应用领域研究人员的必备工具。

本章将围绕 MATLAB 数学形态学图像处理操作的内容，主要是二值图像的处理，并包括简要灰度图像的形态学操作，介绍图像处理工具箱中对于实现数学形态学操作的函数应用方法。

9.1 数学形态学图像处理基础

9.1.1 数学形态学简介

数学形态学（Mathematical Morphology）是法国和德国的科学家在研究岩石结构时建立的一门学科。形态学的用途主要是获取物体拓扑和结构信息，它通过物体和结构元素相互作用的某些运算，得到物体更本质的形态。近年来数学形态学在数字图像处理和机器视觉领域中得到了广泛的应用，形成了一种独特的数字图像分析方法和理论。其基本思想是利用一个称作结构元素（structuring element）的“探针”收集图像信息。当探针在图像中不断移动时，便可考察图像各个部分之间的相互关系，从而了解图像各个部分的结构特征。作为“探针”的结构元素，可直接携带知识（形态、大小以及灰度和色度信息）来探测所研究图像的结构特点。所以，从某种意义上讲，形态学图像处理是以几何学为基础的，着重于研究图像的几何结构。

数学形态学基于探测的思想与人的视觉特点有类似之处：人总是首先关注一些感兴趣的物体或者结构（比如线结构），并有意识地寻找图像中的这些结构。这在人的视觉研究中称为注意力集中，简称 FOA（Focus Attention）。

数学形态学在图像处理中的应用主要是：（1）利用形态学的基本运算，对图像进行观察和处理，从而达到改善图像质量的目的；（2）描述和定义图像的各种几何参数和特征，如面积、周长、连通度、颗粒度、骨架和方向性等。

首先来定义一些基本符号和概念。

1. 元素

设有一幅图像 X，若点 a 在 X 的区域以内，则称 a 为 X 的元素，记作 $a \in X$，如图 9-1（a）

所示。

2. B 包含于 X

设有两幅图像 B 和 X，对于 B 中的任意一个元素 b，都有 $b \in X$，则称 B 包含于(included in)X，记作 $B \subset X$，如图 9-1（b）所示。

3. B 击中 X

设有两幅图像 B 和 X，若存在这样一个点，它既是 B 的元素，又是 X 的元素，且有 $B \not\subset X$，则称 B 击中(hit)X，记作 $B \uparrow X$，如图 9-1（c）所示。

4. B 未击中 X

设有两幅图像 B，X。若不存在任何一个点，它既是 B 的元素，又是 X 的元素，即 B 和 X 的交集是空，则称 B 未击中(miss)X，记作 $B \cap X = \Phi$；其中 $\cap$ 是集合运算相交的符号，Φ 表示空集，如图 9-1（d）所示。

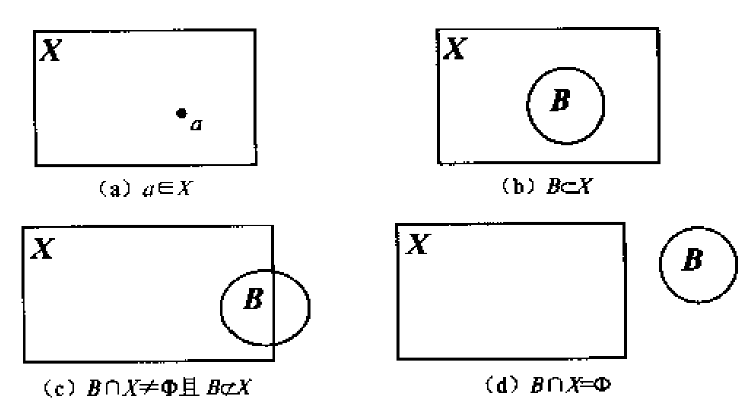


图 9-1 元素与集合之间、几何与几何之间的关系

5. 补集

设有一幅图像 X，所有 X 区域以外的点构成的集合称为 X 的补集，记作 X^c，如图 9-2 所示。显然，如果 $B \cap X = \Phi$，则 B 在 X 的补集内，即 $B \subset X^c$。

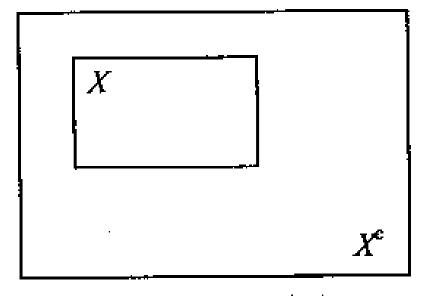


图 9-2 补集的示意图

6. 结构元素

结构元素（structure element），又被形象地称作刷子，是膨胀和腐蚀操作的最基本组成部分，用于测试输入图像，通常要比待处理的图像小得多。二维结构元素由一个数值为 0 或 1 的矩阵组成。结构元素的原点指定了图像中需要处理的像素范围，结构元素中数值为 1 的点决定结构元素的邻域像素在进行膨胀或腐蚀操作时是否需要参与计算。

7. 对称集

设有一幅图像 B，将 B 中所有元素的坐标取反，即令(x, y)变成$(-x, -y)$，所有这些点构成的新的集合称为 B 的对称集，记作 B^v，如图 9-3 所示。

8. 平移

设有一幅图像 B，有一个点 $a(x_0, y_0)$，将 B 平移 a 后的结果是，把 B 中所有元素的横坐标加 x_0，纵坐标加 y_0，即令(x, y)变成$(x+x_0, y+y_0)$，所有这些点构成的新的集合称为 B 的平移，记作 B_a，如图 9-4 所示。

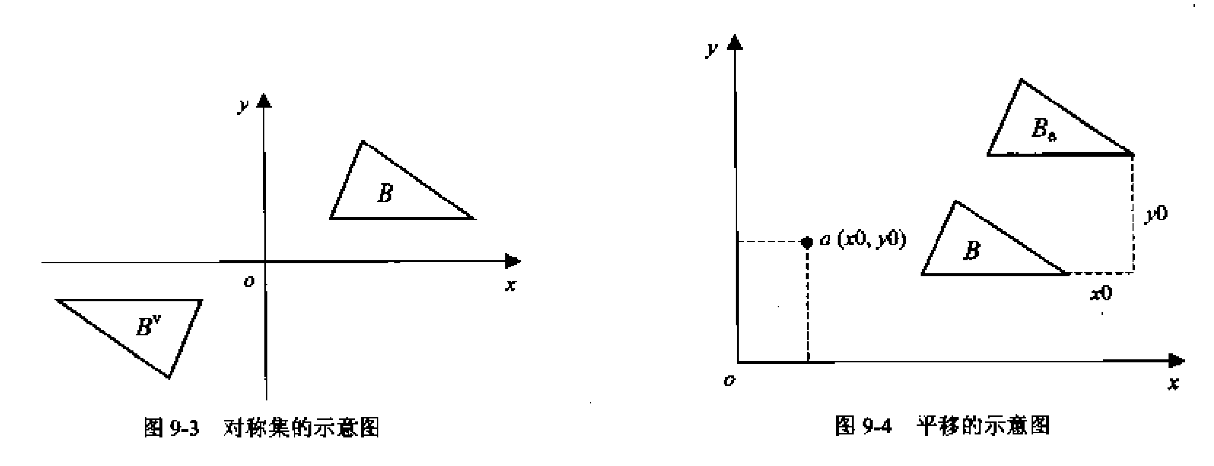


图 9-3　对称集的示意图　　　　图 9-4　平移的示意图

9.1.2　数学形态学的基本运算

1. 腐蚀

把结构元素 B 平移 a 后得到 B_a，若 B_a 包含于 X，我们记下这个 a 点，所有满足上述条件的 a 点组成的集合称作 X 被 B 腐蚀(Erosion)的结果。用公式表示为：$E(X)=\{a|B_a \subset X\}=X\ominus B$，如图 9-5 所示。

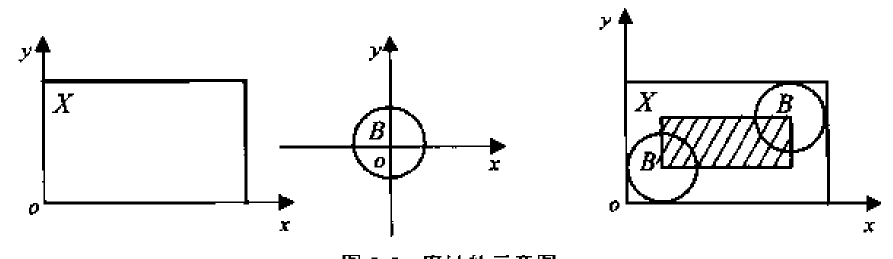


图 9-5　腐蚀的示意图

值得注意的是，上面的 B 是对称的，即 B 的对称集 $B^v=B$，所以 X 被 B 腐蚀的结果和 X 被 B^v 腐蚀的结果是一样的。如果 B 不是对称的，让我们看看图 9-6，就会发现 X 被 B 腐蚀的结果和 X 被 B^v 腐蚀的结果不同。

在图 9-7 中，左边是被处理的图像 X（二值图像，我们针对的是黑点），中间是结构元素 B，那个标有 origin 的点是中心点，即当前处理元素的位置，我们在介绍图像块操作时也有过类似的概念。腐蚀的方法是，移动 B 的中心点和 X 上的点一个一个地对比，如果 B 上的所有点都在 X 的范围内，则该点保留，否则将该点去掉；右边是腐蚀后的结果。可以看出，它仍在原来 X 的范围内，且比 X 包含的点要少，就像 X 被腐蚀掉了一层。

从下面的实例中能够很明显地看出腐蚀的效果，其中图 9-8 为原图，图 9-9 为腐蚀后的结果图。

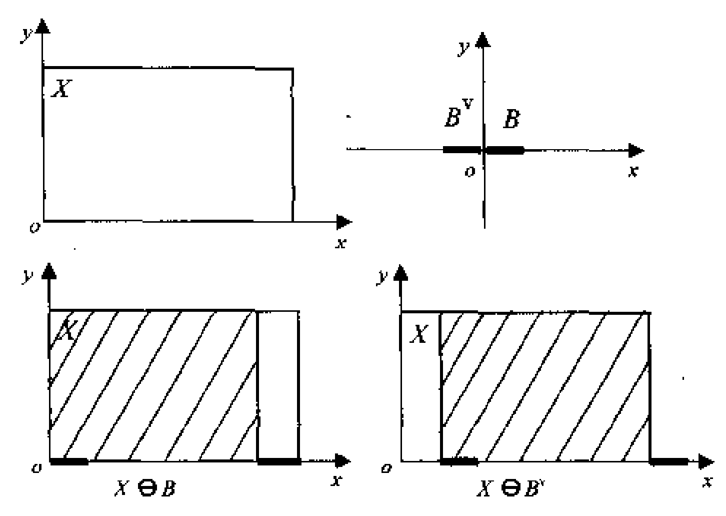


图 9-6 结构元素非对称时，腐蚀的结果不同

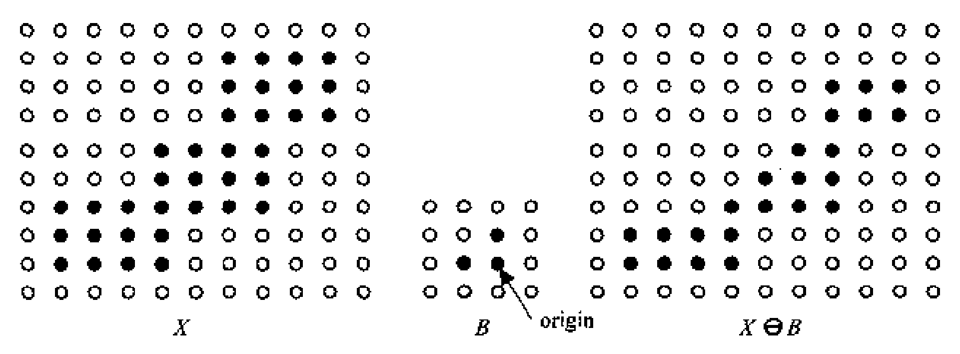


图 9-7 腐蚀运算

Hi,I'm phoenix .
Glad to meet u.

图 9-8 原图

Hi,I'm phoenix .
Glad to meet u.

图 9-9 腐蚀后的结果图

2. 膨胀

膨胀（dilation）可以看作是腐蚀的对偶运算，其定义是：把结构元素 B 平移 a 后得到 B_a，若 B_a 击中 X，我们记下这个 a 点。所有满足上述条件的 a 点组成的集合称作 X 被 B 膨胀的结果。用公式表示为：$D(X)=\{a|B_a\uparrow X\}=X\oplus B$，如图 9-10 所示。图 9-10 中 X 是被处理的对象，B 是结构元素，不难知道，对于任意一个在阴影部分的点 a，B_a 击中 X，所以 X 被 B 膨胀的结果就是那个阴影部分。阴影部分包括 X 的所有范围，就像 X 膨胀了一圈似的，这就是为什么叫膨胀的原因。

同样，如果 B 不是对称的，X 被 B 膨胀的结果和 X 被 B^v 膨胀的结果不同。

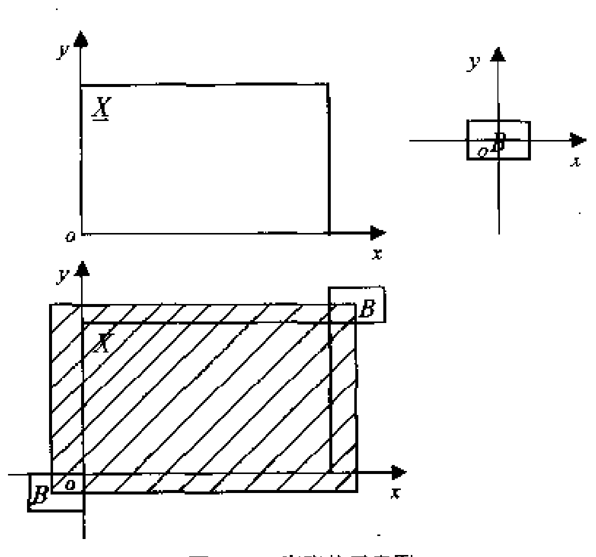


图 9-10 膨胀的示意图

让我们来看看实际上是怎样进行膨胀运算的。在图 9-11 中，左边是被处理的图像 X（二值图像，我们针对的是黑点），中间是结构元素 B。膨胀的方法是，移动 B 的中心点和 X 上的点及 X 周围的点一个一个对比，如果 B 上有一个点落在 X 的范围内，则该点就为黑；右边是膨胀后的结果。如图 9-12 所示为对图 9-8 进行图像膨胀运算的结果。

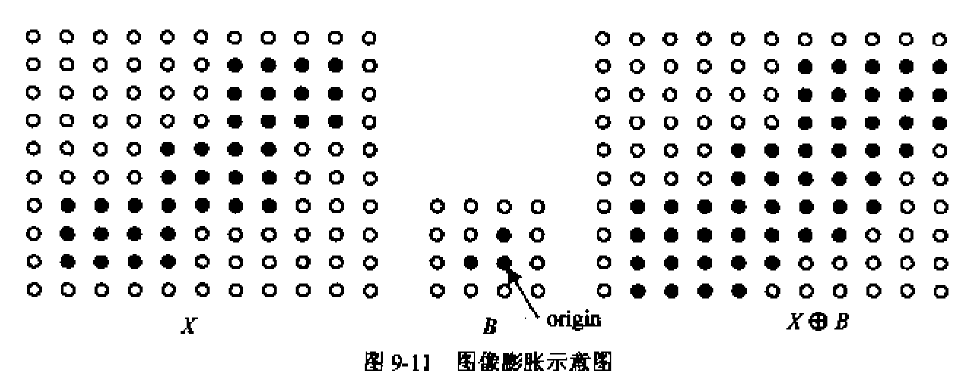


图 9-11 图像膨胀示意图

Hi,I'm phoenix .
Glad to meet u.

图 9-12 对图 9-8 进行图像膨胀运算的结果

3. 开运算和闭运算

在定义腐蚀和膨胀运算的基础上，可以定义数学形态学的另外两个常用运算：开运算（open）和闭运算（close）。

先腐蚀后膨胀称为开，即 OPEN(X)=D(E(X))。

先膨胀后腐蚀称为闭，即 CLOSE(X)=E(D(X))。

开和闭这两种运算可以除去比结构元素小的特定图像细节，同时保证不产生全局几何失真。开运算可以把比结构元素小的突刺滤掉，切断细长搭接而起到分离作用；闭运算可以把比结构元素小的缺口或孔填充上，搭接短的间断而起到连接作用。

让我们来看一个开运算的例子（见图 9-13）。

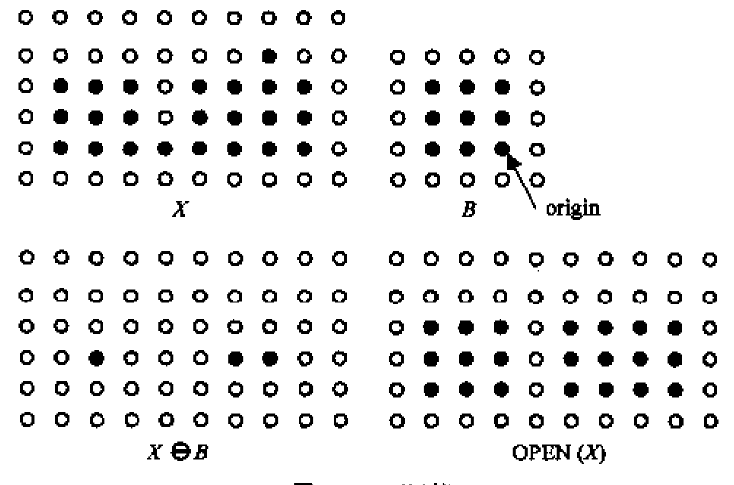


图 9-13 开运算

在该图上面的两幅图中，左边是被处理的图像 X（二值图像，我们针对的是黑点），右边是结构元素 B；下面的两幅图中左边是腐蚀后的结果，右边是在此基础上膨胀的结果。可以看到，原图经过开运算后，一些孤立的小点被去掉了。一般来说，开运算能够去除孤立的小点、毛刺和小桥（即连通两块区域的小点），而总的位置和形状不变。这就是开运算的作用。

要注意的是，如果 B 是非对称的，进行开运算时要用 B 的对称集 B^{v} 膨胀，否则，开运算的结果和原图相比要发生平移，图 9-14 和图 9-15 能够说明这个问题。

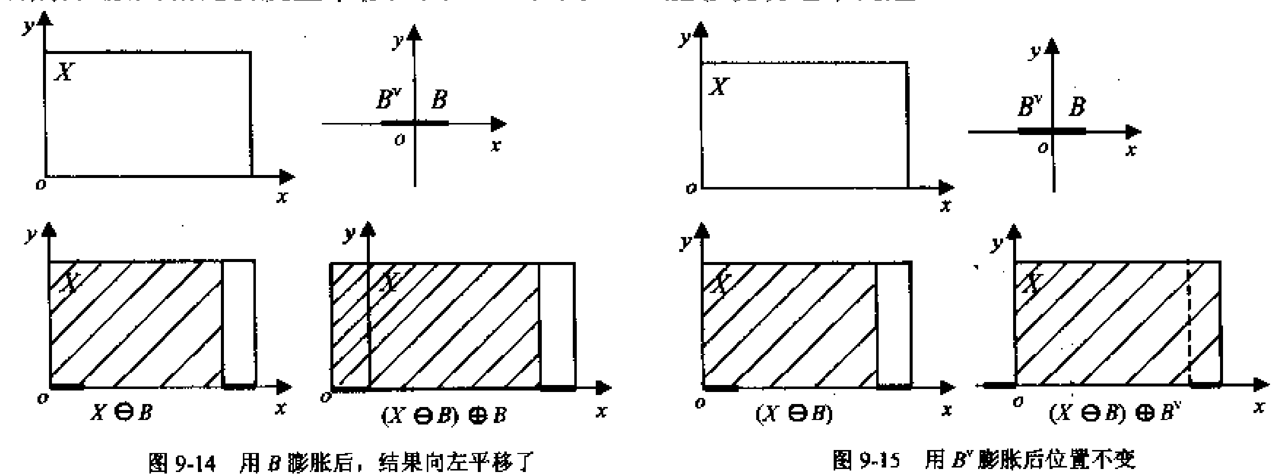


图 9-14 用 B 膨胀后，结果向左平移了

图 9-15 用 B^{v} 膨胀后位置不变

图 9-14 是用 B 膨胀的，可以看到，OPEN(X)向左平移了。图 9-15 是用 B^{v} 膨胀的，可以看到，总的位置和形状不变。

Hi,I'm phoenix .
Glad to meet u.

图 9-16 图 9-8 经过开运算后的结果

图 9-16 是对图 9-8 经过开运算后的结果。

下面介绍闭运算的一个例子，如图 9-17 所示。

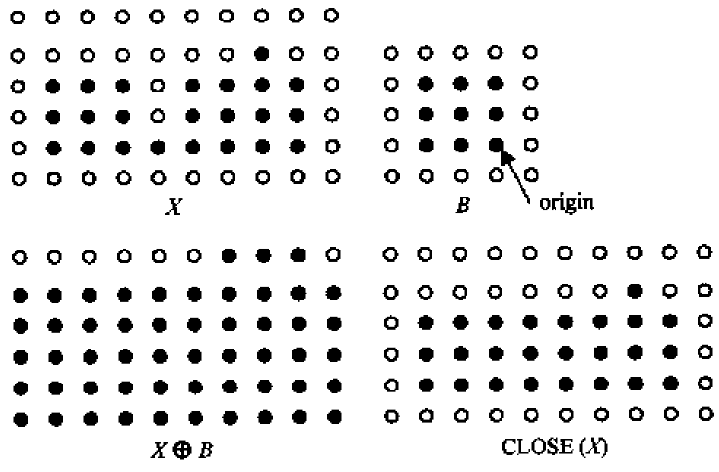


图 9-17 闭运算

在图 9-17 上面的两幅图中，左边是被处理的图像 X（二值图像，我们针对的是黑点），右边是结构元素 B，下面的两幅图中左边是膨胀后的结果，右边是在此基础上腐蚀的结果可以看到，原图经过闭运算后，断裂的地方被弥合了。一般来说，闭运算能够填平小湖（即小孔），弥合小裂缝，而总的位置和形状不变。这就是闭运算的作用。同样要注意的是，如果 B 是非对称的，进行闭运算时要用 B 的对称集 B^{v} 膨胀，否则，闭运算的结果和原图相比要发生平移。如图 9-18 所示为图 9-8 经过闭运算后的结果。

Hi,I'm phoenix .
Glad to meet u.

图 9-18 图 9-8 经过闭运算后的结果

4. 细化

所谓细化（thinning），通俗地讲，细化过程就是对图像逐层剥离的过程，但仍要保持原

图的骨架。所谓骨架，可以理解为图像的中轴，例如一个长方形的骨架是它的长方向上的中轴线；正方形的骨架是它的中心点；圆的骨架是它的圆心，直线的骨架是它自身，孤立点的骨架也是自身。细化一般要遵循一定的原则，其中包括：（1）输出骨架应是一条细线（只有一个像素的宽度）；（2）细化结果是原目标图像的中心线；（3）细化过程中不破坏图像的连通性；（4）具有好的稳定性。

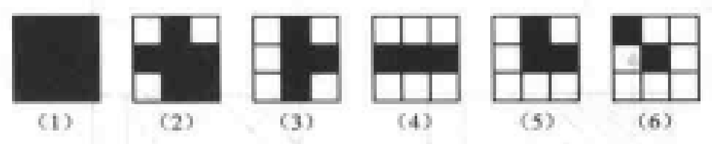


图 9-19　根据某点的 8 个相邻点的情况来判断该点是否能删除

图 9-19 中，（1）的中心点不能删除，因为它是个内部点，我们要求的是骨架，如果连内部点也删除了，骨架也会被掏空；（2）的中心点不能删除，和（1）是同样的道理；（3）的中心点可以删除，这样的点不是骨架；（4）的中心点不能删除，因为删掉后，原来相连的部分断开了；（5）的中心点可以删除，这样的点不是骨架；（6）的中心点不能删除，因为它是直线的端点，如果这样的点删掉了，那么最后整个直线也被删了，剩不下什么。

总结一下，有如下的判据：（1）内部点不能删除；（2）孤立点不能删除；（3）直线端点不能删除；（4）如果 P 是边界点，去掉 P 后，如果连通分量不增加，则 P 可以删除。

我们可以根据上述判据，事先做出一张表，从 0 到 255 共有 256 个元素，每个元素要么是 0，要么是 1。我们根据某点（当然是要处理的黑色点）的 8 个相邻点的情况查表，若表中的元素是 1，则表示该点可删除，否则保留。

9.2　图像膨胀与腐蚀的 MATLAB 实现

9.2.1　结构元素的创建

1. 结构元素的原点

MATLAB 的形态函数使用以下函数获得任意大小和维数的结构元素的原点坐标：

origin = floor((size(nhood)+1)/2)

在上面的语句中，nhood 是指结构元素定义的邻域。结构元素在 MATLAB 中定义为一个 STREL 对象。由于 MATLAB 规定不能在表达式中直接使用对象本身的大小，因此必须使用 STREL 对象的 nhood 属性来获得结构元素的邻域。图 9-20 给出了一个钻石形结构元素的示例。

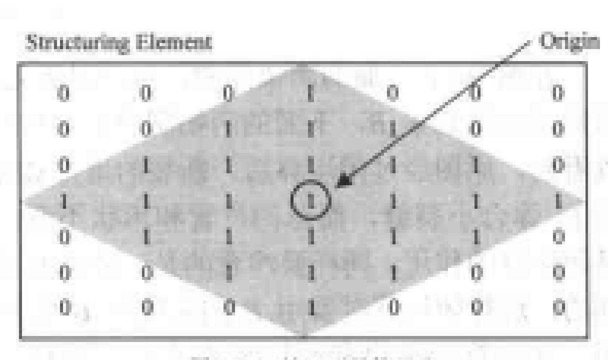


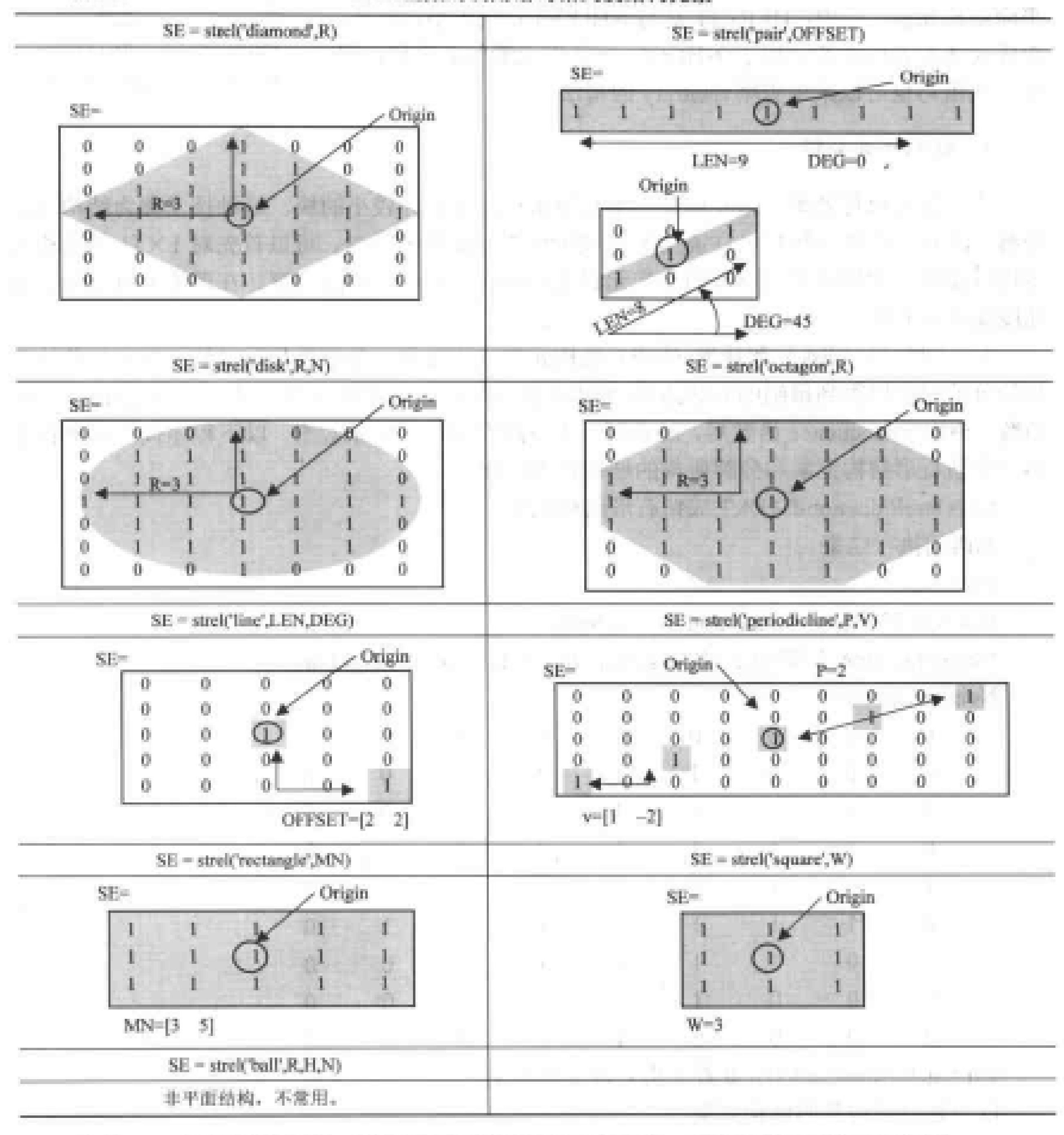
图 9-20　钻石形结构元素

2. 创建结构元素

可以使用 MATLAB 图像处理工具箱中的 strel 函数来创建任意大小和形状的 STREL 对

象。strel 函数支持许多种常用的形状，如线形、钻石形、圆盘形和球形等。如表 9-1 所示列出了 strel 所支持的所有形状的结构元素。

表 9-1　　　　　strel 函数的调用语句格式及结构元素



另外，strel 函数还支持一种任意形状的结构元素创建，即利用如下的语句调用：

SE=strel('arbitrary',NHOOD)

SE=strel('arbitrary',NHOOD,HEIGHT)

语句 SE=strel('arbitrary',NHOOD)用于创建一个平面结构元素，其中 NHOOD 是一个只包含 0 和 1 元素的矩阵，用于表示邻域，NHOOD 的中心（origin）为 floor((size(NHOOD)+1)/2)。另外，简便的写法是省略'arbitrary'，而直接写为 strel(NHOOD)。

例如，执行以下代码可以得到如图 9-21 所示的结构元素。

SE=strel('arbitrary',NHOOD)

执行结果为

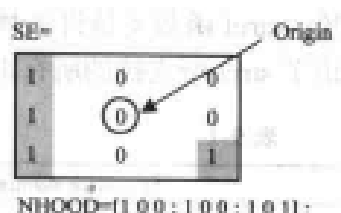


图 9-21 任意形状的结构元素

语句 SE=strel('arbitrary',NHOOD,HEIGHT)用于创建一个非平面结构元素，其中，HEIGHT 是与 NHOOD 相同大小的矩阵，矩阵元素为有限实数，用于 NHOOD 中非零元素的高度值。另外，该语句也可以采用省略'arbitrary'的写法。

3. 结构元素分解

为了提高执行效率，strel 函数可能会将结构元素拆为较小的块，这种技术称为结构元素分解。例如，要对一个 11×11 正方形结构元素进行膨胀操作时，可以首先对 1×11 的结构元素进行膨胀，然后再对 11×1 的结构元素进行膨胀，通过这样的分解，在理论上可以使执行速度提高 6.5 倍。

对于圆盘形（disk）和球形（ball）结构元素进行分解，其结果是近似的，而对于其他形状的分解则可以得到精确的分解结果。要想看到分解所得的结构元素序列，可调用 getsequence 函数。调用 getsequence 函数后，就返回一个分解后的结构元素数组。以下程序代码示例将获得一个钻石形结构元素和分解所得的结构元素序列。

```
sel = strel('diamond',4)%生成钻石形结构元素
%语句执行结果
sel =
Flat STREL object containing 41 neighbors.
Decomposition: 3 STREL objects containing a total of 13 neighbors
Neighborhood:
     0     0     0     0     1     0     0     0     0
     0     0     0     1     1     1     0     0     0
     0     0     1     1     1     1     1     0     0
     0     1     1     1     1     1     1     1     0
     1     1     1     1     1     1     1     1     1
     0     1     1     1     1     1     1     1     0
     0     0     1     1     1     1     1     0     0
     0     0     0     1     1     1     0     0     0
     0     0     0     0     1     0     0     0     0
seq = getsequence(sel)%查看结构元素分解结果
以下是分解得到的结构元素：
seq =
3x1 array of STREL objects
seq(1)%查看分解结构元素 1
执行结果为
ans =
Flat STREL object containing 5 neighbors.
Neighborhood:
```

```
    0     1     0
    1     1     1
    0     1     0
```

seq(2) %查看分解结构元素 2

执行结果为

```
ans =
Flat STREL object containing 4 neighbors.
Neighborhood:
    0     1     0
    1     0     1
    0     1     0
```

seq(3) %查看分解结构元素 3

执行结果为

```
ans =
Flat STREL object containing 4 neighbors.
Neighborhood:
    0     0     1     0     0
    0     0     0     0     0
    1     0     0     0     1
    0     0     0     0     0
    0     0     1     0     0
```

9.2.2 图像膨胀函数

MATLAB 图像处理工具箱提供用于图像膨胀的函数是 imdilate，其调用格式为：

```
IM2 = imdilate(IM,SE)
IM2 = imdilate(IM,NHOOD)
IM2 = imdilate(IM,SE,PACKOPT)
IM2 = imdilate(…,PADOPT)
```

以上语句表示对输入图像 IM 进行按指定结构元素（SE 或 NHOOD）的膨胀操作，得到输出图像 IM2。其中，SE 是由函数 strel 得到的结构元素；NHOOD 是一个只包含 0 和 1 作为元素值的矩阵，用于表示自定义形状的结构元素；PACKOPT 和 PADOPT 是两个优化因子，分别可以取值"ispacked"、"notpacked" 和 "same"、"full"，用来指定输入图像是否为压缩的二值图像和输出图像的大小。

9.2.3 图像腐蚀函数

MATLAB 图像处理工具箱提供用于图像压缩的函数是 imerode，其调用格式为：

```
IM2 = imerode(IM,SE)
IM2 = imerode(IM,NHOOD)
```

IM2 = imerode(IM,SE,PACKOPT,M)

IM2 = imerode(…,PADOPT)

以上语句表示对输入图像 IM 进行按指定结构元素（SE 或 NHOOD）的腐蚀操作，得到输出图像 IM2。其中，SE 是由函数 strel 得到的结构元素；NHOOD 是一个只包含 0 和 1 作为元素值的矩阵，用于表示自定义形状的结构元素；PACKOPT 和 PADOPT 是两个优化因子，分别可以取值“ispacked”、“notpacked”和“same”、“full”，用来指定输入图像是否为压缩的二值图像和指定输出图像的大小。

9.2.4 基于膨胀与腐蚀的形态学操作函数

前面介绍了 MATLAB 中用于图像膨胀和图像腐蚀的工具箱函数，本节将介绍以图像膨胀和图像腐蚀为基础的形态操作函数。重点介绍函数 bwmorph。

通过改变参数，使用该函数可以实现很多类型的图像形态学操作。其调用格式为：

BW2 = bwmorph (BW, operation)

BW2 = bwmorph (BW, operation, n)

这些语句是对输入图像 BW 执行由参数 operation 指定的形态学运算，n 表示反复执行的次数，可以取无穷大，当执行结果不再改变时停止运行。表 9-2 列出了函数 bwmorph 的参数 operation 的可选值。

表 9-2　　函数 bwmorph 支持的形态学操作类型

Operation	描　述
'bothat'	执行形态学的闭包运算，即先执行闭运算（先膨胀后腐蚀），然后减去原图像。
'bridge'	对未连接的像素搭桥，即如果一个 0 值像素两边有非 0 值邻域，则将此 0 值像素置为 1。例如: 1 0 0　　　　1 1 0 1 0 1　变为　1 1 1 0 0 1　　　　0 1 1
'clean'	移除孤立像素点，即 0 0 0　　　　0 0 0 0 1 0　变为　0 0 0 0 0 0　　　　0 0 0
'close'	执行形态学的闭运算（先膨胀后腐蚀）
'diag'	采用对角线填充来去除 8 邻域的背景。例如 0 1 0　　　　0 1 0 1 0 0　变为　1 1 0 0 0 0　　　　0 0 0
'dilate'	用结构元素 ones(3)来执行膨胀运算
'erode'	用结构元素 ones(3)来执行腐蚀运算
'fill'	填充孤立的内部像点，例如： 1 1 1　　　　1 1 1 1 0 1　变为　1 1 1 1 1 1　　　　1 1 1
'hbreak'	移除 H 连接的像素点，例如： 1 1 1　　　　1 1 1 0 1 0　变为　0 0 0 1 1 1　　　　1 1 1
'majority'	如果在某像素的 3×3 邻域中值为 1 的像素点数大于等于 5 个时，将该像素的值置为 1，否则置为 0

续表

Operation	描　述
'open'	执行形态学的开运算（先腐蚀后膨胀）
'remove'	移除内部像素点。如果某像素的4邻域都为1，则将该像素值置为0，这样就只剩下了边界像素点
'shrink'	n为无穷大，反复做收缩运算，把没有孔的目标图像块收缩为一个点，而把含有孔的目标图像块收缩为一个相连的环，环的位置在每个孔内外边缘的中间，收缩运算保持欧拉数不变
'skel'	n为无穷大，反复移除目标图像的边界像素，但不允许原本连接的目标图像断裂，剩下的像素就组成了图像的骨架。这个操作保持欧拉数不变
'spur'	去除目标图像中小的分支像素，例如： 0 0 0 0　　　0 0 0 0 0 0 0 0　　　0 0 0 0 0 0 1 0　变为　0 0 0 0 0 1 0 0　　　0 1 0 0 1 1 0 0　　　1 1 0 0
'thicken'	n为无穷大，反复对目标图像进行粗化操作，即对目标图像的外边缘增加像素，直到原来未连接的多个目标图像按照8邻域被连接起来。粗化操作也保持欧拉数不变
'thin'	n为无穷大，反复对目标图像进行细化操作，即移除目标图像的外边缘像素，直到无孔的目标图像收缩为按照最小程度的接触式连接，而有孔的目标图像按照收缩得到的环进行连接。细化操作也保持欧拉数不变
'tophat'	用开运算后的图像减去原图像

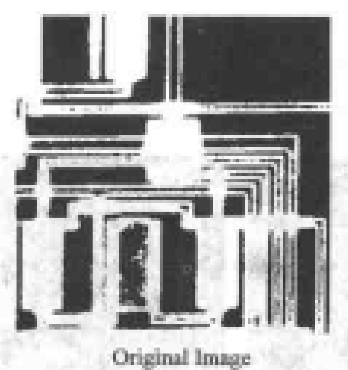
下面介绍在图像处理中广泛应用的骨架化操作和提取边界操作，读者注意区分这里的对二值图像进行的提取边界操作与书中介绍的对灰度图像进行的提取边缘操作是不一样的。这里只涉及图像腐蚀运算，比对灰度图像的提取边缘操作简单得多。

1. 骨架化

在某些应用中，针对一幅图像，希望将图像中的所有对象简化为线条，但不修改图像的基本结构，保留图像的基本轮廓，这个过程就是所谓的骨架化。这里调用函数 bwmorph，可以实现骨架化操作。代码如下：

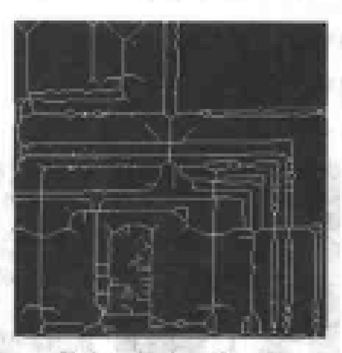
```
BW1 = imread('circbw.tif');
BW2 = bwmorph(BW1,'skel',Inf);
imshow(BW1)
figure, imshow(BW2)
```

处理结构如图 9-22（b）所示。



Original Image

（a）原始输入图像



Skeletonization of Image

（b）骨架化结果图像

图 9-22　图像骨架化处理

2. 提取边界操作

对于一幅二值图像，决定一个像素为边界点，需要满足两个条件：

（1）如果该像素属于原图像区域；

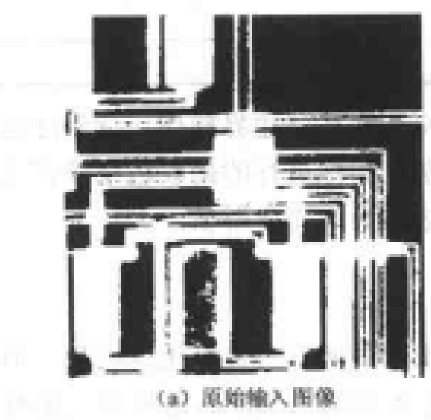
（2）该像素的邻域中至少含有一个不属于原图像区域中的像素点。

注意：这里所说的邻域是指 4-连通邻域，而不是 8-连通邻域。关于像素连通性的概念在后续的内容里还会详细介绍。

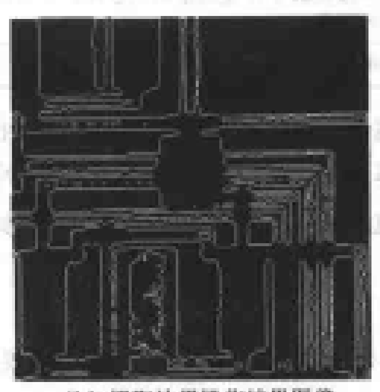
要实现提取边界操作，可以使用 MATLAB 图像处理工具箱提供的专门函数 bwperim。实现代码如下：

```
BW1 = imread('circbw.tif');
BW2 = bwperim(BW1);
imshow(BW1)
figure, imshow(BW2)
```

执行结果如图 9-23 所示。



（a）原始输入图像



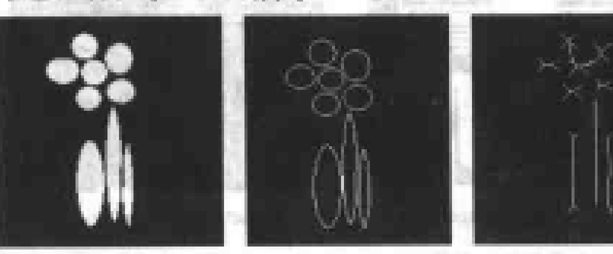
（b）提取边界操作结果图像

图 9-23 对二值图像提取边界操作

另外，也可以调用函数 bwmorph 执行上述提取边界的操作，对于如图 9-24（a）所示的简单图像，进行图像骨架化操作和对图像进行移除内部像素点的操作，代码如下：

```
BW1 = imread('flower.tif');
imshow(BW1)
BW2 = bwmorph(BW1,'remove');%实现提取边界操作
BW3 = bwmorph(BW1,'skel',Inf); %实现骨架化操作
figure, imshow(BW2)
figure, imshow(BW3)
```

得到的结果如图 9-24（b）和（c）所示。



（a）原始图像　（b）提取边界的图像　（c）骨架化图像

图 9-24 使用函数 bwmorph 实现图像的提取边界操作和骨架化操作

除了上面介绍的函数 bwmorph 和 bwperim 外，在 MATLAB 图像处理工具箱中还提供了其他一些基于图像腐蚀和图像膨胀操作的形态学操作函数，如表 9-3 所示。

表 9-3　　基于图像腐蚀和图像膨胀的形态学操作函数

函　数	形态学定义
bwhitmiss	实现二值图像的击中与未击中操作，调用格式为： BW2 = bwhitmiss(BW1,SE1,SE2)，SE1 和 SE2 是两个结构元素，该语句表示保留图像中匹配 SE1 但不匹配 SE2 的像素点。 BW2 = bwhitmiss(BW1,INTERVAL)，INTERVAL 是用单个矩阵合成表示结构元素 SE1 和 SE2，其中，元素为 1 表示 SE1 的对应元素为 1，元素为-1 表示 SE2 的对应元素为 1，其余为 0。 该函数的功能相当于 SE1 对原图像进行腐蚀操作，再用 SE2 对结果图像进行腐蚀操作
imbothat	从执行形态学闭运算的图像中减去原始图像，可以被用于找到图像中的灰度边，功能上类似于利用函数 bwmorph 带参数 bothat，只不过这里处理的是灰度图像，后者处理的是二值图像。函数的调用格式为： IM2 = imbothat(IM,SE) IM2 = imbothat(IM,NHOOD) SE 和 NHOOD 都表示结构元素，前者由函数 strel 得到，后者通过自定义 0-1 矩阵得到。举例如图 9-25 所示
imclose	对灰度图像执行形态学闭运算，即使用同样的结构元素先对图像进行膨胀操作后进行腐蚀操作。调用格式为： IM2 = imclose(IM,SE) IM2 = imclose(IM,NHOOD) 参数说明同上。举例如图 9-26 所示
imopen	对灰度图像执行形态学闭运算，即使用同样的结构元素先对图像进行腐蚀操作后进行膨胀操作。调用格式为： IM2 = imopen(IM,SE) IM2 = imopen(IM,NHOOD) 参数说明同上。举例如图 9-27 所示
imtophat	从执行形态学开运算的图像中减去原始图像，可用来增强图像的对比度。该函数功能上类似于利用函数 bwmorph 带参数 tophat，只不过这里处理的是灰度图像，后者处理的是二值图像。函数的调用格式为： IM2 = imtophat(IM,SE) IM2 = imtophat(IM,NHOOD) 参数说明同上。举例如图 9-28 所示

1. 函数 imbothat

实例代码如下：

```
I = imread('pout.tif');
imshow(I)
se = strel('disk',3);
J = imsubtract(imadd(I,imtophat(I,se)), imbothat(I,se));
figure, imshow(J)
```

执行结果如下图所示。



（a）原始图像



（b）结果图像

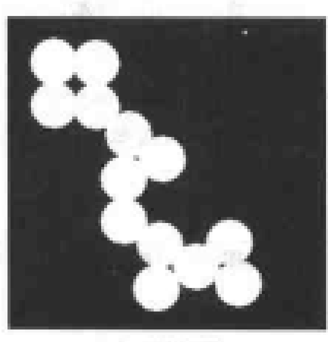
图 9-25　利用函数 imbothat 处理图像

2. 函数 imclose

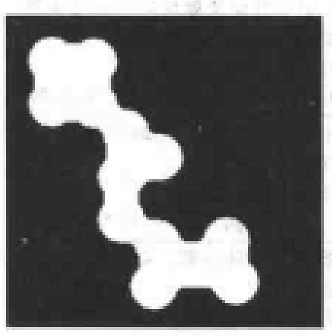
实例代码如下：

```
originalBW = imread('circles.png');
imview(originalBW);
se = strel('disk',10);
closeBW = imclose(originalBW,se);
imview(closeBW)
```

执行结果如图 9-26 所示。



（a）原始图像



（b）结果图像

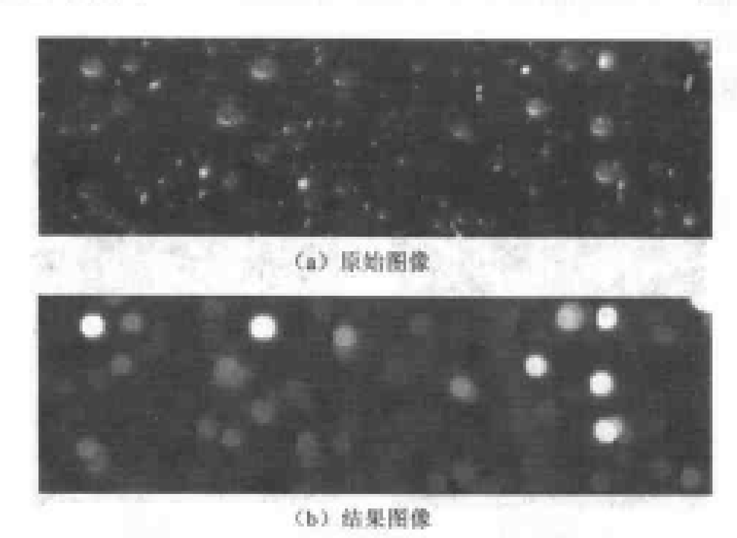
图 9-26 利用函数 imclose 执行图像闭运算

3. 函数 imopen

实例代码如下：

```
I = imread('snowflakes.png');
imview(I)
se = strel('disk',5);
I_opened = imopen(I,se);
imview(I_opened,[])
```

执行结果如图 9-27 所示。



（a）原始图像

（b）结果图像

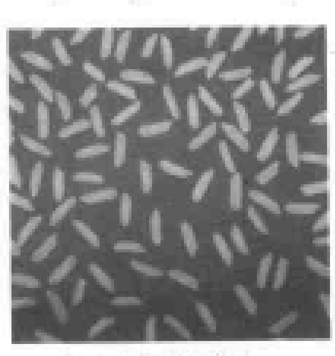
图 9-27 利用函数 imopen 执行图像开运算

4. 函数 imtophat

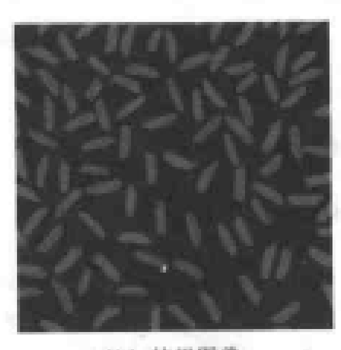
实例代码如下：

```
I = imread('rice.png');
imshow(I)
se = strel('disk',12);
J = imtophat(I,se);
figure, imshow(J)
```

处理结果如图 9-28 所示。



（a）原始图像



（b）结果图像

图 9-28 利用函数 imtophat 增强图像对比度

9.3 形态学重建

所谓形态学重建就是根据一幅图像（称之为掩模图像）的特征对另一幅图像（称之为标记图像）进行重复膨胀操作，直到该图像的像素值不再变化为止。形态学重建是图像形态处理的重要操作之一，通常用来强调图像中与掩模图像指定对象相一致的部分，同时忽略图像中的其他对象。形态学重建有如下三个属性：

（1）处理过程是基于两幅图像——标记图像和掩模图像，而不是一幅图像和一个结构元素；

（2）处理过程反复进行，直到处理结果稳定，例如图像不再变化；

（3）处理过程是基于连通性的概念，而不是基于结构元素。

9.3.1 标记图像和掩模图像

形态学重建是基于掩模图像处理标记图像的过程，标记图像中的峰值点是处理过程的起始点，该过程持续进行，直到标记图像中的像素值不再变化。下面举例说明形态学重建的方法。其中 A 表示掩模图像矩阵，该图像包含两个主要区域，即包含像素值为 14 和 18 的两个图像块，背景像素值大部分被设置为 10，还有一些是 11，如图 9-29 所示。

重建过程可以分为两步：

（1）创建标记图像

与膨胀和腐蚀操作中使用结构元素类似，标记图像的特征决定了形态学重建结果所具有

的特征，所以标记图像的峰值应该确定掩模图像中希望强调对象的位置。在 MATLAB7.0 中，一种创建标记图像的方法就是使用函数 imsubtract 将掩模图像减去一个常数，结果如图 9-30 所示。例如：

```
A = [10   10   10   10   10   10   10   10   10   10;
     10   14   14   14   10   10   11   10   11   10;
     10   14   14   14   10   10   10   11   10   10;
     10   14   14   14   10   10   11   10   11   10;
     10   10   10   10   10   10   10   10   10   10;
     10   11   10   10   10   18   18   18   10   10;
     10   10   10   11   10   18   18   18   10   10;
     10   10   11   10   10   18   18   18   10   10;
     10   11   10   11   10   10   10   10   10   10;
     10   10   10   10   10   10   11   10   10   10];
```

图 9-29　形态学重建的掩模图像矩阵

```
mask=A;
marker=imsubtract(mask, 2);
```

```
marker=
     8    8    8    8    8    8    8    8    8    8
     8   12   12   12    8    8    9    8    9    8
     8   12   12   12    8    8    8    9    8    8
     8   12   12   12    8    8    9    8    9    8
     8    8    8    8    8    8    8    8    8    8
     8    9    8    8    8   16   16   16    8    8
     8    8    8    9    8   16   16   16    8    8
     8    8    9    8    8   16   16   16    8    8
     8    9    8    9    8    8    8    8    8    8
     8    8    8    8    8    8    9    8    8    8
```

图 9-30　生成的标记图像

（2）调用函数 imreconstruct 进行图像形态学重建，调用以下语句，处理结果如图 9-31 所示。

```
recon = imreconstruct(marker, mask)
```

```
recon=
    10   10   10   10   10   10   10   10   10   10
    10   12   12   12   10   10   10   10   10   10
    10   12   12   12   10   10   10   10   10   10
    10   12   12   12   10   10   10   10   10   10
    10   10   10   10   10   10   10   10   10   10
    10   10   10   10   10   16   16   16   10   10
    10   10   10   10   10   16   16   16   10   10
    10   10   10   10   10   16   16   16   10   10
    10   10   10   10   10   10   10   10   10   10
    10   10   10   10   10   10   10   10   10   10
```

图 9-31　标记图像形态学重建的结果

形态学重建从概念上可以被认为是反复膨胀标记图像，直到标记图像的轮廓与掩模图像的轮廓相符。图 9-32 说明了一维图像形态学重建（连续膨胀）的过程。从图中可以看出，每一次连续的膨胀都被限制在掩模以下。当进一步迭代不再改变图像时，过程终止。最终的膨胀结果就是被重建的图像，如图 9-33 所示。

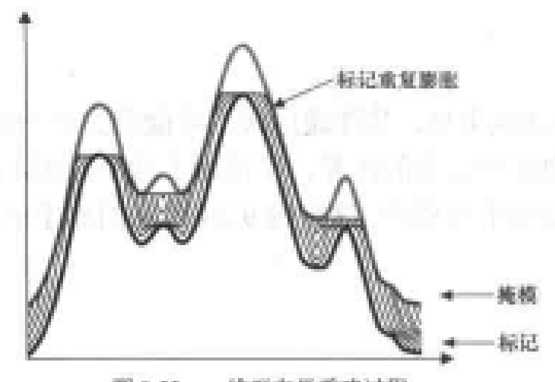


图 9-32 一维形态学重建过程

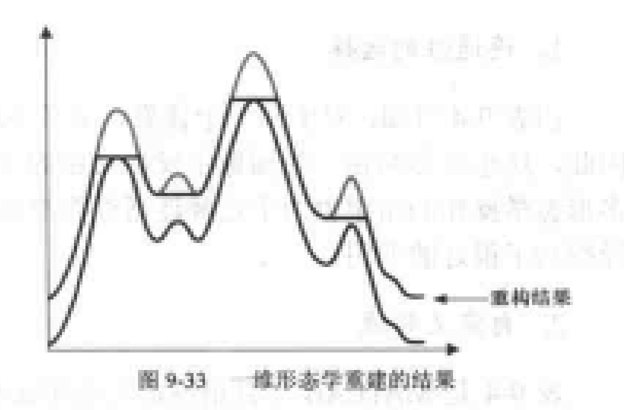


图 9-33 一维形态学重建的结果

9.3.2 像素的连通性

像素的连通性定义了与该像素相连的其他像素。如图 9-34 所示，该二值图像中包含了一个前景目标，该目标对应着值为 1 的像素点。如果前景是 4-连通的，那么图像就只有一个背景块，其所有像素值为 0；如果前景是 8-连通的，则前景就被认为是组成了一个闭环，该图像中就有了两个分离的背景块，分别位于环的内外。

```
0 0 0 0 0 0 0 0
0 1 1 1 1 1 0 0
0 1 0 0 0 1 0 0
0 1 0 0 0 1 0 0
0 1 0 0 0 1 0 0
0 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

图 9-34 像素连通性示例

表 9-4 列出了所有工具箱支持的标准二维和三维连通类型。其中，前两种为二维连通类型，后三种为三维连通类型。

表 9-4 工具箱支持的标准二维和三维连通类型

连通类型	说明	图例
4-连通	边界接触的像素，即只有水平和垂直方向相邻的像素才被认为是属于同一对象	
8-连通	接触相邻和对角相连的像素，即水平、垂直或对角方向相邻的像素都被视为同一对象	
6-连通	面相连的像素	6 faces
18-连通	面或边相连的像素	
26-连通	面、边或角相连的像素	

1. 连通性的选择

由表 9-4 可知，对于同一个像素，定义不同的邻域类型，其连通形式也可能会完全不同。因此，这也会影响在一幅图像中发现对象的个数和这些对象的边界。正是由于这个原因，许多形态学操作的结果也由于选择连通性类型的不同而不尽相同。这在图 9-34 所示的例子中已经得到了很好的证明。

2. 自定义邻域

表 9-4 是 MATLAB 工具箱预定义的邻域连通类型。当然，用户也可以根据需要自定义连通类型。自定义邻域由一个元素数值为 0 或 1 的 3×3×……×3 数组来指定，数值 1 指定相对于中心元素的邻域连通性。例如，使用以下数组，可以定义一个能够将图像分割为列，具有纵向连通性的邻域，即只有处于中心像素垂直方向上的像素才被认为是属于该中心像素的邻域。

```
CONN =
      0     1     0
      0     1     0
      0     1     0
```

9.3.3 区域填充操作

在 MATLAB 工具箱中，函数 imfill 用于执行二值图像和灰度图像的区域填充操作。对于二值图像，imfill 将连接的背景像素（值为 0）变成前景像素（值为 1），当达到图像对象边界时操作停止；对于灰度图像，imfill 将被亮区域围绕的暗区域赋予和亮区域同样的灰度值。区域填充操作可以通过两个步骤实现：

（1）指定区域操作的连通性；

（2）对于二值图像的填充，指定起始点。

函数 imfill 的调用格式为：

```
BW2 = imfill(BW,locations,CONN)
BW2 = imfill(BW,CONN,'holes')
I2 = imfill(I,CONN)
BW2 = imfill(BW)
[BW2 locations] = imfill(BW)
```

其中，BW 和 I 分别表示待处理的二值图像和灰度图像；locations 表示执行二值图像区域填充的起始点位置，一般用两个元素的数组表示；CONN 指定了连通性，其数值可选 4、8、6、18 和 26，分别对应于表 9-4 中所列的 CONN-连通类型，默认值为 4。

BW2 = imfill(BW,CONN,'holes')用于填充二值图像 BW 中的孔（像素值为 0 的背景，该孔的边界不能达到图像边界）；I2 = imfill(I,CONN)用于填充灰度图像中的孔（被亮区域包围的暗区域）；BW2 = imfill(BW)表示允许利用鼠标点击交互式地选取起始点，当执行这一语句后，二值图像 BW 显示在屏幕上，用户可以在指定起始点处点击鼠标左键获得起始点，如果选取不当，可以通过按 Backspace 或 Delete 键撤销选定点，要得到最终的起始点需要移动点击、右击或双击鼠标来完成，按 Return 键可以不做起始点选择而退出选择。[BW2 locations] =

imfill(BW)可以获取交互方式获得的起始点坐标。

注意：交互式选取起始点的方法只适用于二维图像。

例如，对图 9-34 所示的二值图像执行区域填充操作，指定起始点为(4,3)。调用 imfill 函数实现以下语句：

imfill(BW, [4 3])

由于该种情况下默认背景是 4-连通的，该语句将仅填充闭环内部的区域。执行结果如下：

```
ans =
     0     0     0     0     0     0     0     0
     0     1     1     1     1     1     0     0
     0     1     1     1     1     1     0     0
     0     1     1     1     1     1     0     0
     0     1     1     1     1     1     0     0
     0     1     1     1     1     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
```

注意：当前所说的连通性是指背景的连通性，即像素值为 0 的像素点的连通性；而不同于针对图 9-34 所讨论的针对前景的连通性，即像素值为 1 的像素点的连通性。

如果指定的起始点相同，但是使用了 8-连通邻域类型，则 imfill 函数将填充整幅图像：

imfill(BW, [4 3], 8)

执行结果如下：

```
ans =
     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1
```

9.3.4 寻找灰度图像的灰度极值点

如果将灰度图像视为三维图像，其中用 x 坐标和 y 坐标来描述像素位置，而 z 坐标表示像素的灰度值。在这种表示方式下，灰度值就类似于地形图中的高程值，图像的高灰度值和低灰度值就相当于地形图的峰和谷。

通常，图像的高低灰度值（极值）代表该图像的相关对象，具有重要的形态特征。因而，使用形态学操作对图像的极值进行处理可以辨识图像中的对象。

1. 极大值和极小值

图像中可能有多个局部极大和极小值，但是只有一个全局的极大和极小值。通过判断图

像的峰和谷，可以创建用于形态学重建的标记图像。图 9-35 表示了一幅一维图像（或二维图像中的一条直线）中的极值分布情况，说明了局部极值和全局极值的关系。

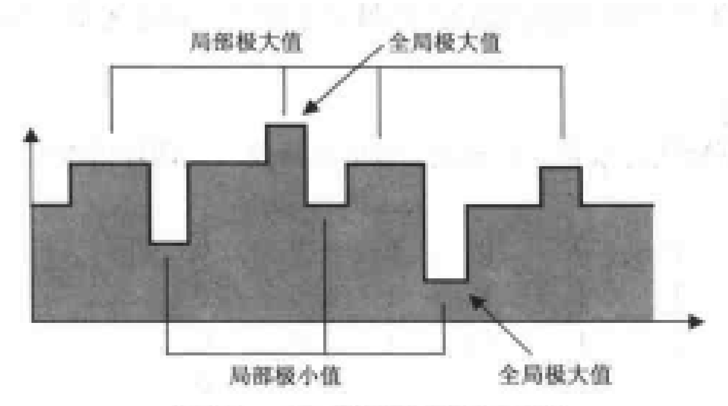


图 9-35　一幅一维图像的极值分布情况

2. 寻找极大和极小灰度值区域

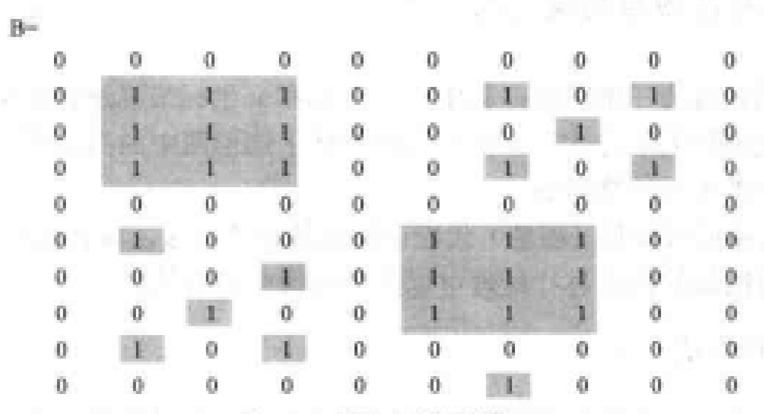
工具箱提供了函数 imregionalmax 和 imregionalmin 分别用于指定灰度值局部极大区域和局部极小区域，函数 imextendedmax 和 imextendedmin 分别用于指定灰度值大于某一阈值的局部极大区域和小于某一阈值的局部极小区域。这些函数接受灰度图像作为输入，输出为标定相关区域的二值图像（极大或极小区域像素值置为 1，其他的像素值为 0）。例如，对于灰度图像 A，其中包含有两个局部极大值区域（值为 13 和 18）和一些小的极值区域（值为 11），即图中阴影所示像素区域，如图 9-36 所示。

```
A = [10   10   10   10   10   10   10   10   10   10;
     10   13   13   13   10   10   11   10   11   10;
     10   13   13   13   10   10   10   11   10   10;
     10   13   13   13   10   10   11   10   11   10;
     10   10   10   10   10   10   10   10   10   10;
     10   11   10   10   10   18   18   18   10   10;
     10   10   10   11   10   18   18   18   10   10;
     10   10   11   10   10   18   18   18   10   10;
     10   11   10   11   10   10   10   10   10   10;
     10   10   10   10   10   10   11   10   10   10];
```

图 9-36　灰度图像 A

调用函数 imregionalmax 找出图像中所有的局部极大值区域，执行结果如图 9-37 所示。

B = imregionalmax(A)

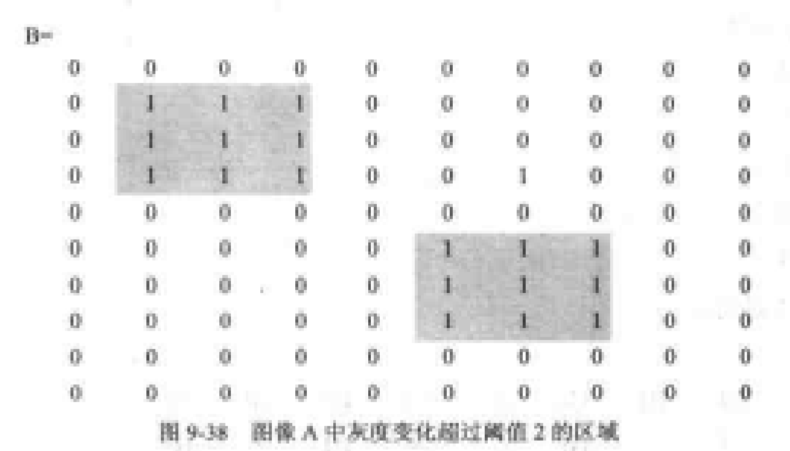


```
B=
     0    0    0    0    0    0    0    0    0    0
     0    1    1    1    0    0    1    0    1    0
     0    1    1    1    0    0    0    1    0    0
     0    1    1    1    0    0    1    0    1    0
     0    0    0    0    0    0    0    0    0    0
     0    1    0    0    0    1    1    1    0    0
     0    0    0    1    0    1    1    1    0    0
     0    0    1    0    0    1    1    1    0    0
     0    1    0    1    0    0    0    0    0    0
     0    0    0    0    0    0    1    0    0    0
```

图 9-37　返回二值结果图像

如果仅仅需要找到那些灰度值变化较大的区域，也就是找出灰度变化大于某一阈值的区域，可调用 imextendedmax 函数来实现，操作结果如图 9-38 所示。

B = imextendedmax(A,2)%表示灰度变化超过阈值 2 的区域



```
B=
     0     0     0     0     0     0     0     0     0     0
     0     1     1     1     0     0     0     0     0     0
     0     1     1     1     0     0     0     0     0     0
     0     1     1     1     0     0     1     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     1     1     1     0     0
     0     0     0     0     0     1     1     1     0     0
     0     0     0     0     0     1     1     1     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
```

图 9-38　图像 A 中灰度变化超过阈值 2 的区域

3. 抑制极大值和极小值

在一幅图像中，灰度的每一个波动都代表了一个局部极值区域。但在多数情况下，我们只对某些显著的极值区域感兴趣，而需要忽略由于背景纹理导致的较小的极值区域。因此，在应用中，要抑制这些小的极值区域，同时又要保持显著的极值区域不变。

要实现这一操作，可以调用工具箱函数 imhmax 或 imhmin。在执行这两个函数时，需要指定一个阈值 h，从而抑制所有灰度值大于 h 的极大值区域或小于 h 的极小值区域。调用函数 imhmax 处理如图 9-29 所示的灰度图像，指定阈值为 2，调用语句格式如下：

B=imhmax(A, 2)

得到的输出图像 B 如图 9-31 所示。可见，尽管极值区域的像素值发生了改变，但该操作却很好地得到了指定阈值变化范围的极大值区域。

注意：函数 imregionalmin、imregionalmax、imextendedmax 和 imextendmin 函数会返回一幅标记了局部极大值或极小值区域的二值图像，而函数 imhmax 和 imhmin 返回的是一幅对原始图像修改后的灰度图像。图 9-39 给出了用函数 imhmax 对第二行进行修改的过程示意，极大值都减小，而极小值保持不变。

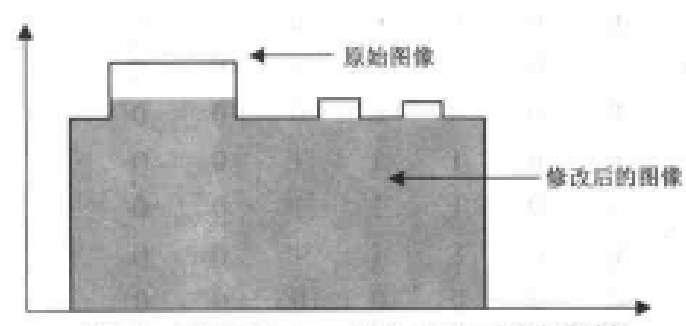


图 9-39　利用函数 imhmax 对图 9-36 第二行的操作过程

4. 突出极小值

利用函数 imimposemin 可以强调图像中特定的极小值区域（暗对象）。该函数可以利用形态学重建的方式消除指定极小值区域以外的所有其他极小值区域。下面的例子给出了突出极小值区

域的过程，下面的代码生成了包含两个主要的局部极小值区域和几个其他局部极小值区域。

```
mask = uint8(10*ones(10,10));
mask(6:8,6:8) = 2;
mask(2:4,2:4) = 7;
mask(3,3) = 5;
mask(2,9) = 9
mask(3,8) = 9
mask(9,2) = 9
mask(8,3) = 9
```

执行结果如下所示。

```
mask =
    10   10   10   10   10   10   10   10   10   10
    10    7    7    7   10   10   10   10    9   10
    10    7    5    7   10   10   10    9   10   10
    10    7    7    7   10   10   10   10   10   10
    10   10   10   10   10   10   10   10   10   10
    10   10   10   10   10    2    2    2   10   10
    10   10   10   10   10    2    2    2   10   10
    10   10    9   10   10    2    2    2   10   10
    10    9   10   10   10   10   10   10   10   10
    10   10   10   10   10   10   10   10   10   10
```

突出极小值区域的过程如下：

（1）创建一幅标记图像。可使用 imextendedmin 函数来获得一幅指明两个极小值位置的二值图像，如下所示。

```
marker = imextendedmin(mask,1)
marker =
     0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0
     0    0    1    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    1    1    1    0    0
     0    0    0    0    0    1    1    1    0    0
     0    0    0    0    0    1    1    1    0    0
     0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0
```

（2）调用函数 imimposemin 来生成掩模图像中由标记图像 marker 指定点的新极小值，这个新的极小值为图像数据类型支持的最小值，如对于 uint8 类型，该极小值为 0。另外，函数 imimposemin 也改变图像中所有其他的像素值来消除其他的极小值。

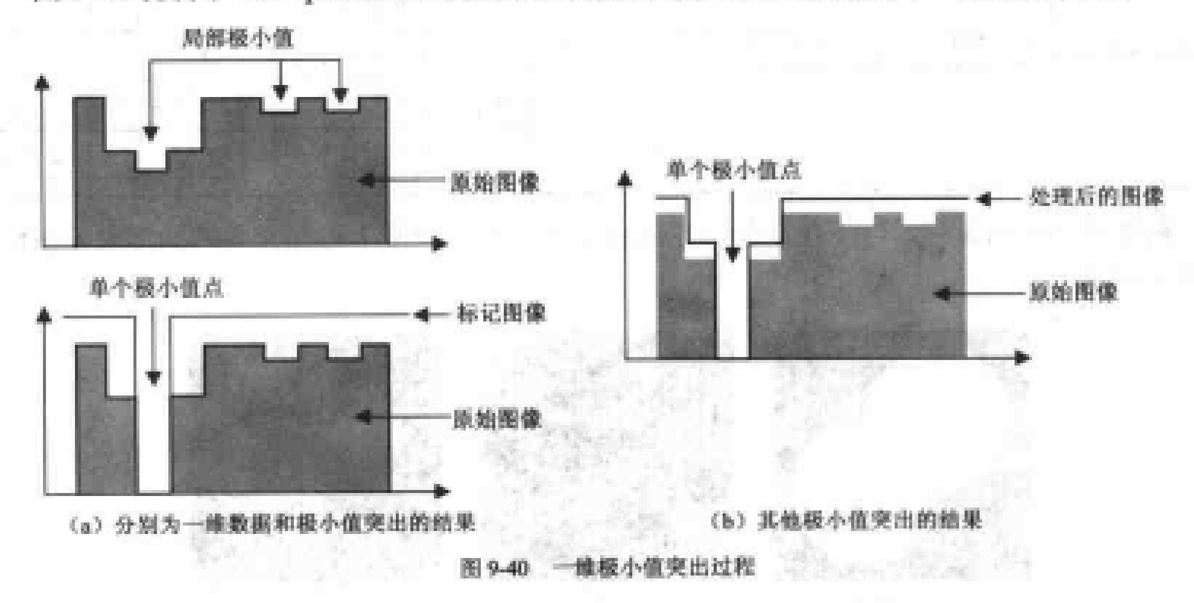
```
I = imimposemin(mask,marker)
```

执行结果如下。

I =

```
    11    11    11    11    11    11    11    11    11    11
    11     8     8     8    11    11    11    11    11    11
    11     8     0     8    11    11    11    11    11    11
    11     8     8     8    11    11    11    11    11    11
    11    11    11    11    11    11    11    11    11    11
    11    11    11    11    11     0     0     0    11    11
    11    11    11    11    11     0     0     0    11    11
    11    11    11    11    11     0     0     0    11    11
    11    11    11    11    11    11    11    11    11    11
    11    11    11    11    11    11    11    11    11    11
```

图 9-40 说明了 imimposemin 函数是如何修改图像中第二行的整个一维执行过程。



（a）分别为一维数据和极小值突出的结果　　（b）其他极小值突出的结果

图 9-40　一维极小值突出过程

9.4 距离变换

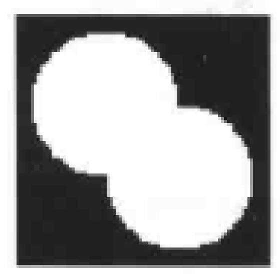
距离变换提供了对图像中像素点分离程度的一个度量。图像处理工具箱中提供了函数 bwdist 用于计算二值图像中每个值为 0 的像素点到最近的非 0 值像素点的距离，并用该距离值作为生成图像的灰度值，如表 9-5 所示。

表 9-5　函数 bwdist 支持的距离类型

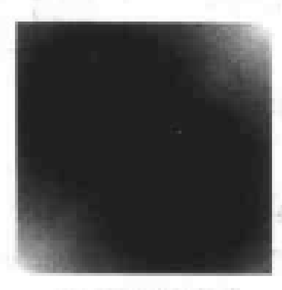
距离表示	含　义	图例（左侧为图像，右侧为距离变换）
Euclidean	两个像素点间的直线距离	0 0 0 / 0 1 0 / 0 0 0 → 1.41 1.0 1.41 / 1.0 0.0 1.0 / 1.41 1.0 1.41

距离表示	含　义	图例（左侧为图像，右侧为距离变换）
City Block	用于度量 4-连通邻域的两个像素点间的距离。边界相连的像素点距离为 1，对角相连的距离为 2	0 0 0 0 1 0 0 0 0 2 1 2 1 0 1 2 1 2
Chessboard	用于度量 8-连通邻域的两个像素点间的距离。边界相连或角相连的像素间距离为 1	0 0 0 0 1 0 0 0 0 1 1 1 1 1 1 1 1 1
Quasi-Euclidean	用于度量沿一组水平、垂直和对角线段的总的欧氏距离(Euclidean distance)	0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 2.8 2.2 2.0 2.2 2.8 2.2 1.4 1.0 1.4 2.2 2.0 1.0 0 1.0 2.0 2.2 1.4 1.0 1.4 2.2 2.8 2.2 2.0 2.2 2.8

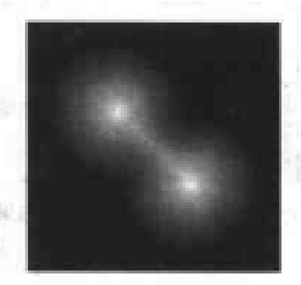
根据表 9-5 的说明，以下程序代码示例将建立一个包含两个相互交叠的圆形对象图像，如图 9-41 所示。



（a）原图像



（b）距离变换图像



（c）距离变换图像灰度取反

图 9-41　距离变换

```
center1 = -10;
center2 = -center1;
dist = sqrt(2*(2*center1)^2);
radius = dist/2 * 1.4;
lims = [floor(center1-1.2*radius) ceil(center2+1.2*radius)];
[x,y] = meshgrid(lims(1):lims(2));
bw1 = sqrt((x-center1).^2 + (y-center1).^2) <= radius;
bw2 = sqrt((x-center2).^2 + (y-center2).^2) <= radius;
bw = bw1 | bw2;
figure, imshow(bw)
D = bwdist(bw);%生成距离变换图像
figure, imshow(D,[])
```

```
D1 = bwdist(~bw); %生成灰度取反的距离变换图像
figure, imshow(D1,[])
```

9.5 区域、对象及特性度量

9.5.1 连通区域标记

图像处理工具箱中提供了函数 bwlabel 和 bwlabeln 用于执行二值图像的连通区域标记操作。其中，bwlabel 只支持 2-D 输入图像，bwlabeln 可以支持任意维数的输入，返回一个与输入图像大小相同的标定矩阵，以不同的整数值来区分输入图像中的不同对象。函数 bwlabel 的调用格式如下：

```
L = bwlabel(BW,n)
[L,num] = bwlabel(BW,n)
```

其中，BW 表示二值输入图像，n 表示像素的连通性，默认值是 8。num 表示在图像 BW 中找到的连通区域数目。

假设有如下二值图像 BW：

```
BW = [1   1   1   0   0   0   0   0
      1   1   1   0   1   1   0   0
      1   1   1   0   1   1   0   0
      1   1   1   0   0   0   1   0
      1   1   1   0   0   0   1   0
      1   1   1   0   0   0   1   0
      1   1   1   0   0   1   1   0
      1   1   1   0   0   0   0   0];
```

在该图像中，值为 1 的像素给定了几个不同的区域。调用函数 bwlabel，同时指定像素的连通性为 4-连通，即

```
L = bwlabel(BW,4)
```

执行结果如下：

```
L =
     1   1   1   0   0   0   0   0
     1   1   1   0   2   2   0   0
     1   1   1   0   2   2   0   0
     1   1   1   0   0   0   3   0
     1   1   1   0   0   0   3   0
     1   1   1   0   0   0   3   0
     1   1   1   0   0   3   3   0
     1   1   1   0   0   0   0   0
```

从图像 L 中可以看出，函数 bwlabel 在原图像中标记了三个区域，分别对应值为 1、2 和

3 的像素区域。若指定像素的连通性为默认的 8-连通，即

L1 = bwlabel(BW)

执行结果如下：

```
L1 =
     1     1     1     0     0     0     0     0
     1     1     1     0     2     2     0     0
     1     1     1     0     2     2     0     0
     1     1     1     0     0     0     2     0
     1     1     1     0     0     0     2     0
     1     1     1     0     0     0     2     0
     1     1     1     0     0     2     2     0
     1     1     1     0     0     0     0     0
```

由图像 L1 可以看出，原本在 4-连通定义下的区域 2 和区域 3 被合并成为一个区域，因为在 8-连通的定义下，这两个区域是连通的，即被认为是一个区域。

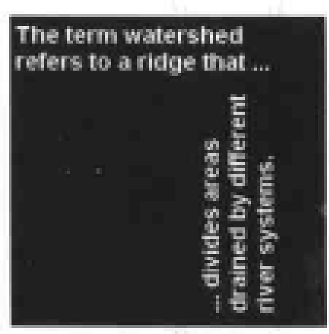
由于 bwlabel 函数的输出矩阵不是二值图像，而是 double 类型的矩阵，因此可以用索引色图像的格式显示该输出矩阵。在显示时，首先将各元素加 1，使各个像素值处于索引色图像的有效像素值范围内。这样就可以调用函数 label2rgb 将每个对象显示为不同的颜色，很容易将各个物体区分开来。例如以下程序代码：

```
x = bwlabel(BW, 4);
RGB = label2rgb(x, @jet, 'k');
imshow(RGB+1, 'notruesize')
```

9.5.2 选择对象

在二值图像中，所谓对象就是指像素值为 1 的像素组成的图像区域。当只对图像中的特定对象感兴趣时，我们通常希望可以只对感兴趣对象进行相关操作。为了完成对象选择操作，图像处理工具箱提供了函数 bwselect。在进行对象选择时，首先需要指定一些像素，然后 bwselect 函数才会返回包含指定像素的二值图像对象。

利用 bwselect 函数可以实现对图像的特征提取操作，例如提取文本图像中的某些字符对象。以下程序代码就是对如图 9-42（a）所示的包含字符的二值图像提取指定的字符，其执行结果如图 9-42（b）所示。



（a） 输入二值图像



（b）含有被选择对象的输出图像

图 9-42 利用函数 bwselect 选择字符对象

```
BW1 = imread('text.png');
c = [43 185 212];
r = [38 68 181];
BW2 = bwselect(BW1,c,r,4);
imview(BW1), imview(BW2)
```

9.5.3 计算图像面积

计算二值图像前景（值为 1 的像素点组成的区域）的面积可以使用工具箱函数 bwarea，该面积可以简单地理解为是图像中像素值为 1 的像素的数目。然而，函数 bwarea 并不是简单地计算图像对象的像素数目，而是为不同的像素模型赋予不同的权值，加权求和得到目标图像的面积。加权操作是为了补偿用离散像素表示连续图像的变形。例如，50 个像素组成的对角线图像就比同样有 50 个像素组成的水平或垂直线图像的面积计算结果要大。水平或垂直情况的面积可以用 50 表示其图像面积，由于加权的操作，对角线图像的面积为 62.5。下面的例子是利用函数 bwarea 来计算对图像 circbw.tif 执行膨胀操作后面积增长的百分比。

```
BW = imread('circbw.tif');
SE = ones(5);
BW2 = imdilate(BW,SE);
increase = (bwarea(BW2) - bwarea(BW))/bwarea(BW);
```

得到的计算结果为：

```
increase = 0.3456
```

函数 bwarea 是通过对图像中所有单个像素面积求和得到总面积的，而单个像素的面积是根据其 2×2 邻域得到的。对于单个像素面积的计算有 6 种不同的模式：

- 邻域中无“1”值像素点的，面积为 0；
- 邻域中有 1 个“1”值像素点的，面积为 1/4；
- 邻域中有 2 个相连的“1”值像素点的，面积为 1/2；
- 邻域中有 2 个对角相连的“1”值像素点的，面积为 3/4；
- 邻域中有 3 个“1”值像素点的，面积为 7/8；
- 邻域中 4 个像素值都为“1”的，面积为 1。

记住图像中的每一个“1”值像素点都可以看作是 4 个不同的 2×2 邻域的一部分，因而，对于无相连或对角相连的“1”值像素点，其总面积为 1。例如，计算二值图像 BW 的面积：

```
0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0
0   0   1   1   0   0   0   0
0   0   1   1   0   0   0   0
0   0   0   0   0   0   0   0
0   0   0   0   0   1   0   0
0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0
```

area=bwarea(BW)%计算图像 BW 的面积

计算结果为 5。

9.5.4 欧拉数

在对图像拓扑进行估计时，经常会使用到一个量——欧拉数。所谓欧拉数，就是一幅图像中的对象数目减去该图像中的孔洞的数目。在模式识别中，经常利用欧拉数进行聚类分析，该分析方法是一种非常有效的方法。在图像处理工具箱中提供了函数 bweuler 用于计算二值图像的欧拉数。

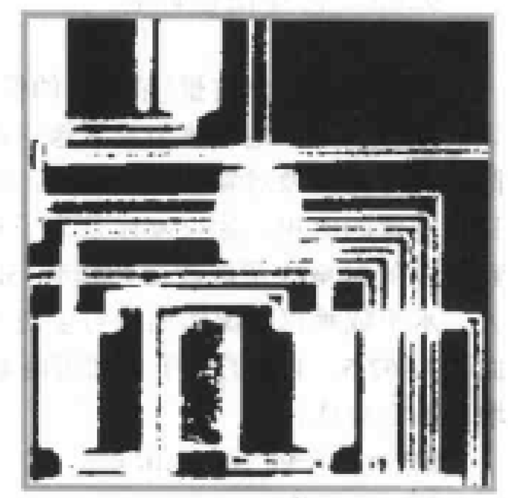


图 9-43 二值图像 circbw.tif

利用函数 bweuler 进行欧拉数计算时，该函数只支持 4-连通和 8-连通两种邻域连通类型。以下程序代码示例说明了计算欧拉数的方法，得到了如图 9-43 所示图像在 8-连通邻域方式下的欧拉数。

```
BW1 = imread('circbw.tif');
eul = bweuler(BW1,8)
```

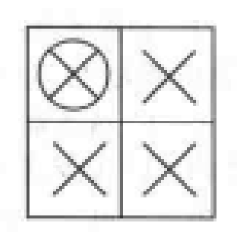
得到的计算结果为：

```
eul =
   -85
```

9.6 查表操作

在 MATLAB 中，一些二值图像操作如果采用查找表（lookup table），可能会提高计算速度。查找表是一个列向量，它把一个像素邻域点的所有可能组合保存起来，使得大量的运算转换为查表问题。

图像处理工具箱提供了函数 makelut 用来产生 2×2 和 3×3 的邻域查找表。一旦创建好查找表，就可以调用函数 applylut，借助所创建的表来完成需要实现的操作。下面是函数中定义的 2×2 和 3×3 邻域。对于 2×2 邻域，总共有 16 种排列方式，因此生成的 2×2 查找表是一个拥有 16 个元素的矢量；对于 3×3 邻域，总共有 512 种排列方式，因此生成的 3×3 查找表是一个拥有 512 个元素的矢量。所有的邻域点都用“×”表示，中心像素点用一个圆圈表示，如图 9-44 所示。



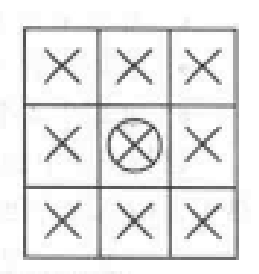


图 9-44 2×2 和 3×3 邻域

1. makelut

函数 makelut 的语法格式为：

```
LUT = makelut(FUN,N)
```

此语句返回函数 FUN 定义的查找表，N 为邻域尺寸，N=2 或者 N=3。

LUT = makelut(FUN,N,P1,P2,⋯)

此语句中 P1,P2,⋯也是传给 FUN 的参数。

makelut 函数生成查找表 LUT 的方法是：把 2×2 或 3×3 邻域内的所有值都传给 FUN，每次一个，构造含 16 个元素的矢量（2×2 邻域）或含 512 个元素的矢量（3×3 邻域）。矢量中包含了 FUN 对每个可能的邻域的输出。

例如，要生成一个 2×2 邻域的查找表，在该例中，当邻域中 1 的个数大于等于 2 时函数返回值为 1，否则返回 0。

```
f = inline('sum(x(:)) >= 2');
lut = makelut(f,2);
```

得到的 2×2 邻域的查找表为：

$lut=[0\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1]^T$

这里$[\,]^T$表示转置。

2．applylut

函数 applylut 用于执行基于查找表的二值图像处理，其语法结构是：

A = applylut(BW,LUT)

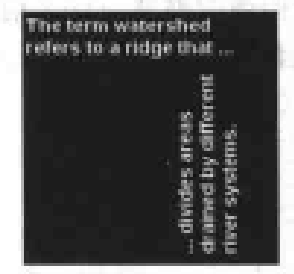
其中 BW 是输入的二值图像，可以是 uint8 型或 double 型；LUT 是由 makelut 返回的 16 元素或 512 元素的矢量，可以是 uint8 型或 double 型；A 的值取决于 LUT 中的值，如果 LUT 中的值全部是 1 和 0，则 A 是二值图像。如果 LUT 中的所有元素都是 0 到 255 之间的整数，则不管 LUT 是什么类型，A 都是 uint8 型的；否则是 double 型的。

下面将通过一个示例，说明如何实现对图像进行查表操作。

使用内联函数，当一个 2×2 邻域内有 4 个的像素值为 1，那么该函数将返回 1，否则返回 0。利用该计算函数，调用函数 makelut，生成查找表，最后执行 applylut 函数操作。实现代码如下：

```
lut = makelut('sum(x(:)) == 4',2);
BW = imread('text.png');
BW2 = applylut(BW,lut);
imview(BW), imview(BW2)
```

执行结果如图 9-45 所示。



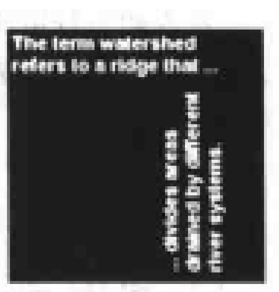


图 9-45　查找表操作示例

第 10 章　图像滤波和滤波器设计

图像滤波是一种通过变更图像的频率域表示而修改或增强图像的技术。例如，我们可以设计一个滤波器，通过对图像中某些特征频率域表示的了解，增强或移除图像中的这些特征。图像滤波操作通常并不是在整幅图像上同时进行，而是一种基于邻域的操作，滤波操作应用特定的算法在此邻域像素上，例如在图像处理中常被用到的掩模操作，如提取边缘的算子（Sobel 算子，Roberts 算子，Prewitt 算子等）、图像平滑以及图像锐化等。

图像滤波本质上就是对输入图像邻域像素的线性加权操作，在图像处理中，它是一种广泛应用的操作方式。在本书的其他章节中，读者已看到图像滤波的大量应用，无论是低通、带通或高通等各种滤波器。然而，读到此处，读者并非十分了解针对特定图像处理任务，应该如何得到所需的滤波器，以及滤波器的参数如何设计。本章将给读者一个答案，让读者掌握如何设计一个合适的图像滤波器。

本章第一节首先介绍图像滤波的计算方法，即卷积和相关计算，然后介绍如何应用一个已有滤波器实现图像滤波操作，并介绍了 MATLAB 工具箱中提供的现有滤波器。第二节介绍了如何根据自身需要，量身定做一个合适的滤波器。

10.1　图像滤波操作

在介绍自行设计图像滤波器方法之前，我们先来了解一下图像滤波的实质，这对读者了解滤波器设计方法有一定帮助，接下来介绍 MATLAB 已有的工具箱函数及其实现滤波的方法。

10.1.1　卷积

已知输入信号 $x(t)$，经过一个线性系统 f，得到输出信号 $y(t)$，这一过程可用下式表示：

$$y(t)=\int_{-\infty}^{\infty}f(t,\tau)x(\tau)\mathrm{d}\tau \tag{10.1}$$

此式一般性地表达了任何线性系统输入 $x(t)$ 和 $y(t)$ 之间的关系。当然，对任何线性系统，必须要选择一个二元函数 $f(t,\tau)$ 使上式成立。然而，我们却希望用一个一元函数来刻画线性系统。

为了简化上式，加入移不变约束条件，即有

$$y(t-T)=\int_{-\infty}^{\infty}f(t,\tau)x(\tau-T)\mathrm{d}\tau \tag{10.2}$$

进行变量替换，将 t 和 τ 同时加上 T，得到

$$y(t)=\int_{-\infty}^{\infty}f(t+T,\tau+T)x(\tau)\mathrm{d}\tau \tag{10.3}$$

由此可得

$$f(t,\tau)=f(t+T,\tau+T) \tag{10.4}$$

要使上式对所有 T 都成立，意味着当两变量增加同样的量时，$f(t,\tau)$ 的值不变，即只要 t 和 τ 的差不变，$f(t,\tau)$ 的函数值也不变。为此，可有如下定义：

$$g(t-\tau)=f(t,\tau) \tag{10.5}$$

从而，有

$$y(t)=\int_{-\infty}^{\infty}g(t-\tau)x(\tau)\mathrm{d}\tau \tag{10.6}$$

上式就表示 $x(t)$和 $g(t)$的卷积运算。该式表明，线性移不变系统的输出可通过输入信号与表征系统特性的函数卷积得到。这个特性函数就称为系统的冲激响应。

以上表征的是一维卷积计算，下面我们来得到二维卷积的定义。假定系统的二维输入信号为 $f(x,y)$，系统的冲激响应函数为 $h(x,y)$，则有输出二维信号为：

$$g(x,y)=h(x,y)*f(x,y)=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}f(u,v)h(x-u,y-v)\mathrm{d}u\mathrm{d}v \tag{10.7}$$

则应用于数字图像处理的离散卷积形式为：

$$g(i,j)=\sum_{m}\sum_{n}f(m,n)h(i-m,j-n) \tag{10.8}$$

若 $h(x,y)$具有可分离性，即有下式：

$$h(x,y)=h_1(x)h_2(y) \tag{10.9}$$

则二维卷积计算可以分成两次一维卷积分别计算：

$$\begin{aligned}g(x,y)=h(x,y)*f(x,y)&=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}f(u,v)h_1(x-u)h_2(y-v)\mathrm{d}u\mathrm{d}v\\&=\int_{-\infty}^{\infty}\left[\int_{-\infty}^{\infty}f(u,v)h_1(x-u)\mathrm{d}u\right]h_2(y-v)\mathrm{d}v\end{aligned} \tag{10.10}$$

其对应的离散形式为：

$$g(i,j)=\sum_{n}\left[\sum_{m}f(m,n)h_1(i-m)\right]h_2(j-n) \tag{10.11}$$

在图像处理中的卷积运算都是针对某像素的邻域进行的，其实质就是对图像邻域像素加权求和得到输出像素值，其中的权矩阵被称为卷积核，也就是图像滤波器。

假如图像矩阵为

```
A = [17  24   1   8  15
     23   5   7  14  16
      4   6  13  20  22
     10  12  19  21   3
     11  18  25   2   9]
```

卷积核为

```
h = [8   1   6
     3   5   7
     4   9   2]
```

如图 10-1 所示，表示利用卷积核 h 计算输出图像像素(2,4)的方法。该方法可分成 4 步：

（1）以卷积核的中心为中心旋转卷积核 180 度；

（2）滑动卷积核的中心到输入图像的像素点(2,4)；

（3）将旋转后的卷积核与像素点(2,4)的邻域像素值对应相乘；

（4）将上一步的乘积求和。

即图像像素(2,4)的值为：1×2+8×9+15×4+16×3+13×6+20×1+22×8=575

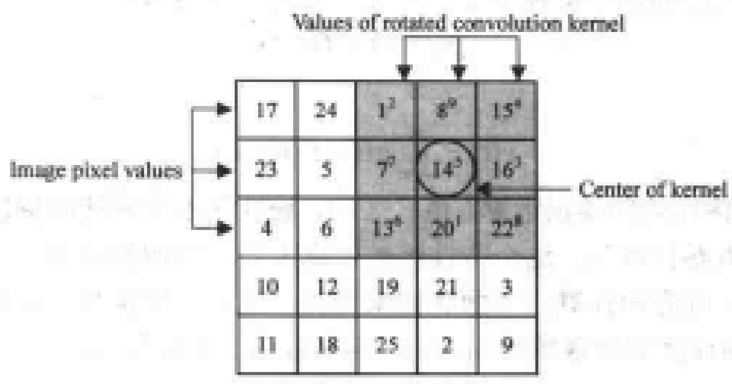


图 10-1　图像卷积计算

卷积定理是线性系统分析中最常用、最重要的定理之一，该定理得到傅立叶变换的一个重要性质：时域中的卷积运算相当于频域中的相乘，即

如果有傅立叶变换关系 $f(t) \Leftrightarrow F(\omega)$，$g(t) \Leftrightarrow G(\omega)$

则有 $f(t) * g(t) \Leftrightarrow F(\omega)G(\omega)$，$f(t)g(t) \Leftrightarrow F(\omega) * G(\omega)$

由此可知，时域中的卷积运算可以在频域中通过简单的乘法来实现：首先，对两个函数进行傅立叶变换，然后对变换结果求乘积，最后求出积的傅立叶变换，就得到了两个函数的卷积结果。如果需要利用卷积进行图像的滤波或其他运算，就可以利用卷积定理，大大减少计算量，因为傅立叶变换可以使用 FFT 计算得到。

10.1.2　相关

两个函数 $f(x,y)$ 与 $g(x,y)$ 的相关性定义如下：

$$f(x,y) \circ g(x,y) = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f^*(m,n)g(x+m,y+n) \tag{10.12}$$

这里 f^* 表示 f 的复共轭。

相关操作与卷积运算比较相似，也是计算邻域像素的加权和，不同之处在于相关运算中的权矩阵被称为相关核，在计算时不需要旋转。如图 10-2 给出了进行相关计算的例子。输入图像仍为 A，相关核为 h。则相关计算包括以下 3 步：

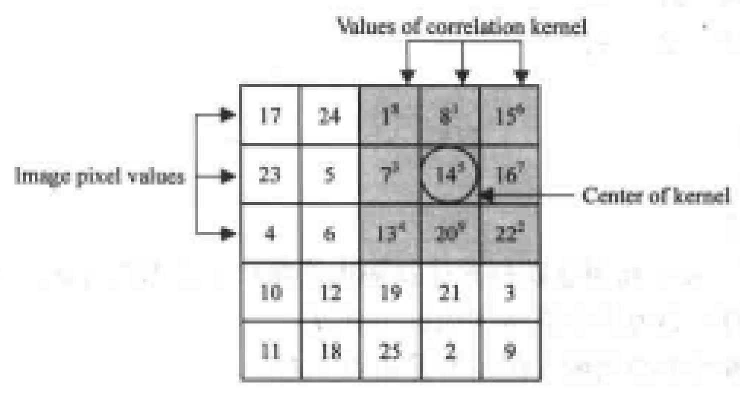


图 10-2　图像相关计算

（1）滑动相关核的中心到图像 A 的像素点(2,4)；

（2）将卷积核与像素点(2,4)的邻域像素对应相乘；

（3）将上一步的乘积求和。

因此，经过相关计算得到的输出像素点(2,4)的像素值为：

$$1*8+8*1+15*6+7*3+14*5+16*7+13*4+20*9+22*2=585$$

对应于卷积定理，相关性定理也是线性系统理论中比较重要的一个定理。由傅立叶变换的性质可以推出以下关系式：

如果有傅立叶变换关系 $f(x,y) \Leftrightarrow F(u,v)$，$g(x,y) \Leftrightarrow G(u,v)$

则有，

$$f(x,y) \circ g(x,y) \Leftrightarrow F^*(u,v)G(u,v)$$

$$f^*(x,y)g(x,y) \Leftrightarrow F(u,v) \circ G(u,v)$$

其中 $F^*(u,v)$表示 $F(u,v)$的复共轭，$f^*(x,y)$表示 $f(x,y)$的复共轭。

由以上定理可知，可以通过计算图像的傅立叶变换，并依此定理，得到图像相关计算的结果。

10.1.3 MATLAB 滤波函数

MATLAB 图像处理工具箱中提供了函数 imfilter 用于图像滤波操作，其调用格式如下：

B = imfilter(A,H)

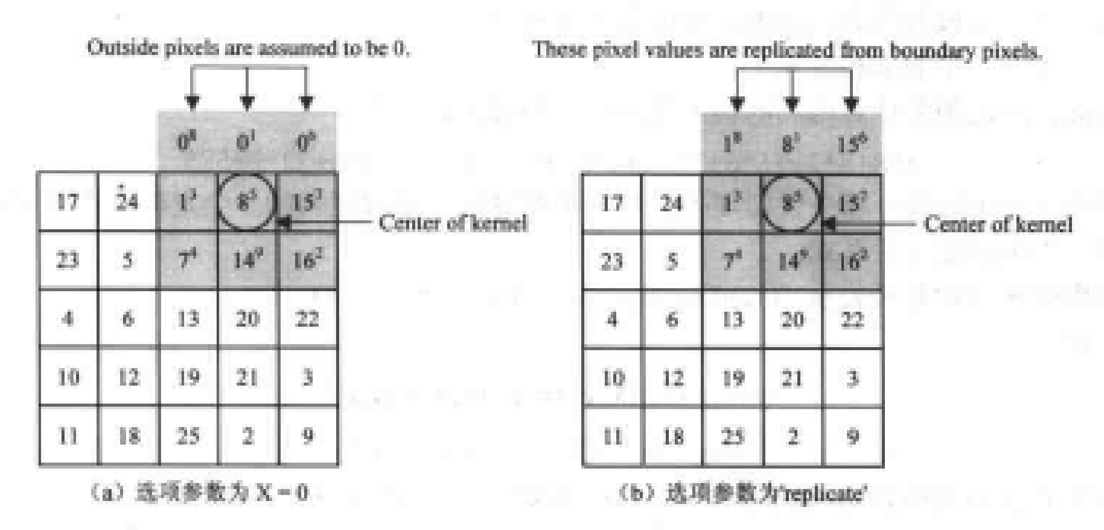
B = imfilter(A,H,option1,option2,⋯)

B = imfilter(A,H)表示利用滤波器 H 对数组 A（或输入图像）进行滤波，得到输出数组（或输出图像）B，这里 A 可以是任意维数和任意类型的数组，输出 B 与 A 的大小和类型相同。B = imfilter(A,H,option1,option2,⋯)表示按照特定的选项 option1,option2,⋯进行多维滤波，选项参数如表 10-1 所示。

表 10-1　　函数 imfilter 的可选参数

边界选项参数	
选　项	含　义
X	数组边界以外的值被隐含地假定为值 X，当未指定 X 时，默认值为 0
'symmetric'	数组边界以外的值通过相对于边界的镜像反射得到
'replicate'	数组边界以外的值被假定为最接近边界的数组元素值
'circular'	数组边界以外的值假定输入数组循环出现
输出数组大小选项参数	
选　项	含　义
'same'	输出数组的大小与输入数组相同，这也是默认的选项
'full'	输出数组是完整的滤波结果，比输入数组要大
相关和卷积选项参数	
选　项	含　义
'corr'	以相关计算进行滤波操作，当未指定该滤波选项参数时，这是默认选项
'conv'	以卷积运算进行滤波操作

图 10-3 分别给出了选取两个不同边界选项（X、'replicate'）的示意图。



（a）选项参数为 X＝0　　（b）选项参数为'replicate'

图 10-3　选取不同边界选项参数的示意图

下面举例说明函数 imfilter 的应用，实现图像平滑滤波的代码如下：

```
I = imread('coins.png');
h = ones(5,5) / 25;
I2 = imfilter(I,h);
imshow(I), title('Original Image');
figure, imshow(I2), title('Filtered Image')
```

处理结果如图 10-4 所示。

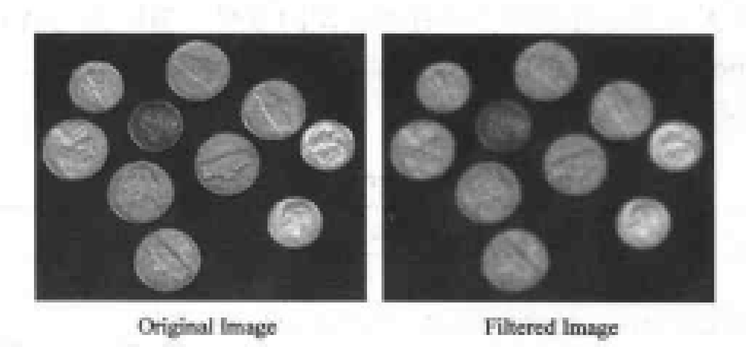

Original Image　　Filtered Image

图 10-4　利用函数 imfilter 进行平滑滤波

10.1.4　使用预定义的滤波器

MATLAB 提供了工具箱函数 fspecial 用来生成几种以相关核的形式表示的预定义滤波器。该滤波器可以用作函数 imfilter 的参数。该函数调用格式如下：

```
h = fspecial(type)
h = fspecial(type,parameters)
```

其中 type 的取值有以下几种选择，见表 10-2。

表 10-2　　函数 fspecial 的参数 type 可选值

type	含　义	Parameters
'gaussian'	高斯低通滤波器	包含两个参数 hsize 和 sigma，其中 hsize 表示 h 的大小，若为标量 n 表明 h 是 n×n 方阵，若为矢量[m n]，表示 h 的行数和列数分别为 n 和 m，默认值为[3 3]；sigma 表示 h 的标准差，默认值 0.5
'sobel'	Sobel 水平边界增强滤波器	无
'prewitt'	Prewitt 水平边界增强滤波器	无
'laplacian'	近似于 2-D Laplacian 算子的滤波器	参数 alpha 用于控制滤波器的形状，取值范围是[0.0,1.0]，默认值为 0.2
'log'	高斯 Laplacian 滤波器	包含两个参数 hsize 和 sigma，其含义与'gaussian'相同，唯一区别是 hsize 的默认值为[5 5]
'average'	均衡滤波器	参数 hsize 含义与'gaussian'型相同
'unsharp'	对比增强滤波器	参数 alpha 与'laplacian'相同
'disk'	圆形均衡滤波器	参数 radius 表示圆形滤波器的半径，表示滤波器为行列数 2*radius+1 的方阵，radius 默认值为 5
'motion'	模拟相机线性运动滤波器	参数 len 和 theta 分别表示运动的像素数和逆时针运动的方向角度数，其默认值分别为 9 和 0

下面分别给出了几种滤波器的计算表达式。

（1）高斯低通滤波器

$$h_g(n_1,n_2)=\mathrm{e}^{-(n_1^2+n_2^2)/2\sigma^2} \tag{10.13}$$

$$h(n_1,n_2)=\frac{h_g(n_1,n_2)}{\sum\limits_{n_1}\sum\limits_{n_2}h_g} \tag{10.14}$$

其中，σ 对应函数参数 sigma。

（2）Laplacian 滤波器

$$\nabla^2=\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2} \tag{10.15}$$

$$\nabla^2\approx\frac{4}{\alpha+1}\begin{bmatrix}\frac{\alpha}{4} & \frac{1-\alpha}{4} & \frac{\alpha}{4}\\ \frac{1-\alpha}{4} & -1 & \frac{1-\alpha}{4}\\ \frac{\alpha}{4} & \frac{1-\alpha}{4} & \frac{\alpha}{4}\end{bmatrix} \tag{10.16}$$

其中，α 对应函数参数 alpha。

（3）高斯 Laplacian 滤波器

$$h_g(n_1,n_2)=\mathrm{e}^{-(n_1^2+n_2^2)/2\sigma^2}$$

$$h(n_1,n_2)=\frac{(n_1^2+n_2^2-2\sigma^2)h_g(n_1,n_2)}{2\pi\sigma^2\sum\limits_{n_1}\sum\limits_{n_2}h_g} \tag{10.17}$$

其中，σ 对应函数参数 sigma。

（4）均衡滤波器

ones(n(1),n(2))/(n(1)*n(2))

（5）对比增强滤波器

$$\frac{1}{\alpha+1}\begin{bmatrix} -\alpha & \alpha-1 & -\alpha \\ \alpha-1 & \alpha+5 & \alpha-1 \\ -\alpha & \alpha-1 & -\alpha \end{bmatrix} \tag{10.18}$$

其中，α 对应函数参数 alpha。

10.2 滤波器设计方法

在 MATLAB7.0 中，滤波器设计都是在频域进行的。本节将在前述内容的基础上，主要介绍下面几种设计方法：

- FIR 滤波器法：图像处理工具箱提供的线性滤波器。
- 频率变换法：即将 1-DFIR 滤波器变换为 2-DFIR 滤波器。
- 频率采样法：创建满足频率响应要求的滤波器。
- 窗口法：将理想脉冲响应与窗函数相乘来创建滤波器。
- 创建满足频率响应要求的矩阵。

10.2.1 FIR 滤波器法

图像处理工具箱中提供了一类线性滤波器，即 2-D 有限脉冲响应滤波器 FIR。FIR 滤波器具有多种特点，是 MATLAB7.0 环境下理想的图像处理工具。这些特点主要包括：

- FIR 滤波器容易用系数矩阵来表示。
- 2-D FIR 滤波器是 1-D FIR 滤波器的自然扩展。
- 有多种成熟的 FIR 滤波器设计方法。
- FIR 滤波器易于实现。
- 可以设计为线性相位，从而避免图像处理中的变形。

无限脉冲响应（IIR）滤波器不适合在图像处理中应用，它在稳定性、设计和实现的容易程度上都不如 FIR 滤波器。因此，MATLAB7.0 图像处理工具箱不支持 IIR 滤波器的应用。

10.2.2 频率变换法

在 MATLAB7.0 中，频率变换法就是将 1-D FIR 滤波器转换为 2-D FIR 滤波器进行设计的方法。该方法的最大特点就是变换后的 2-D FIR 滤波器保留了变换前 1-D FIR 滤波器的大多数属性。在频率变换法中，MATLAB7.0 用到了一个变换矩阵（transformation matrix），该矩阵是一组元素构成的数据集，定义了频率的变换关系。

在 MATLAB7.0 图像处理工具箱中，提供了 ftrans2 函数用于实现频率变换。利用该函数默认的变换矩阵，可以生成一个近似于圆对称的滤波器。当然，也可以自己定制变换矩阵，从而得到各种不同对称性的滤波器。

函数 ftrans2 的调用格式为

h = ftrans2(b,t)

h = ftrans2(b)

其中，b 表示已知的 1-D 滤波器，t 表示使用的变换矩阵。h = ftrans2(b)表示默认使用 McClellan 变换矩阵。

下面的变换过程定义了由函数 ftrans2 得到的 2-D 滤波器的频率响应。假设已知的 1-D 滤波器为 b，其傅立叶变换为 $B(\omega)$，即

$$B(\omega)=\sum_{n=-N}^{N} b(n)\mathrm{e}^{-\mathrm{j}\omega n} \tag{10.19}$$

则得到的变换函数的频率响应为

$$H(\omega_1,\omega_2)=B(\omega)\Big|_{\cos\omega=T(\omega_1,\omega_2)} \tag{10.20}$$

其中，$T(\omega_1,\omega_2)$ 表示变换矩阵 t 的傅立叶变换，即

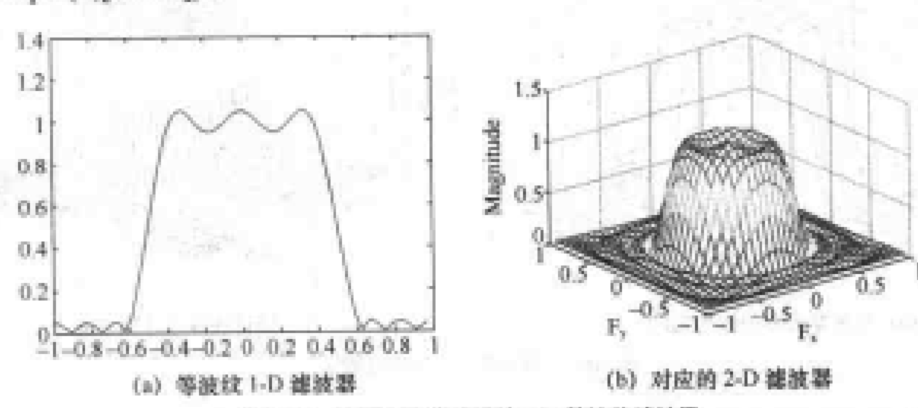
$$T(\omega_1,\omega_2)=\sum_{n_2}\sum_{n_1} t(n_1,n_2)\mathrm{e}^{-\mathrm{j}\omega_1 n_1}\mathrm{e}^{-\mathrm{j}\omega_2 n_2} \tag{10.21}$$

则得到对应的 2-D 滤波器 h 为 $H(\omega_1,\omega_2)$ 的傅立叶反变换，即

$$h(n_1,n_2)=\frac{1}{(2\pi)^2}\int_{-\pi}^{\pi}\int_{-\pi}^{\pi} H(\omega_1,\omega_2)\mathrm{e}^{\mathrm{j}\omega_1 n_1}\mathrm{e}^{\mathrm{j}\omega_2 n_2}\mathrm{d}\omega_1\mathrm{d}\omega_2 \tag{10.22}$$

频率变换法通常可以得到很好的结果，因为设计具有特定属性的 1-D 滤波器要比设计相对应的 2-D 滤波器更加容易。以下的程序代码实例就给出了这样一个设计过程，首先生成一个优化的等波纹 1-D 滤波器，然后利用频率变换法，生成一个对应的 2-D 滤波器，其特性如图 10-5 所示。

```
b = firpm(10,[0 0.4 0.6 1],[1 1 0 0]);
h = ftrans2(b);
[H,w] = freqz(b,1,64,'whole');%得到 b 的 1-D 频率响应
colormap(jet(64))%图形颜色映射
plot(w/pi-1,fftshift(abs(H)))
figure, freqz2(h,[32 32])
```



(a) 等波纹 1-D 滤波器　　(b) 对应的 2-D 滤波器

图 10-5　用频率变换法设计 2-D 等波纹滤波器

firpm 用于指定等波纹滤波器的设计参数，对于调用格式 B=firpm(N,F,A)，返回值 B 表示一个长度 N+1 的线性相位（实对称系数）FIR 滤波器，参数 F 和 A 描述了该滤波器的频率响应。其中，F 是一个成对的频带边界矢量，其值在[0,1]范围内按升序排列；1 对应于奈奎斯特

频率或采样频率的一半；A 是与 F 相同大小的实矢量，指定了滤波器 B 的频率响应的幅度。(F(k),A(k))就对应了频率响应的特征点，按照等波纹滤波器的特性，连接这些点并依据对称性就得到了等波纹滤波器 B 的频率响应。

函数 freqz2 用于计算 2-D 滤波器的频率响应特性，对于该函数的调用格式 freqz2(h,[n2 n1])，h 表示一个 2-D 滤波器，n2 和 n1 指定了频率响应矩阵的大小。

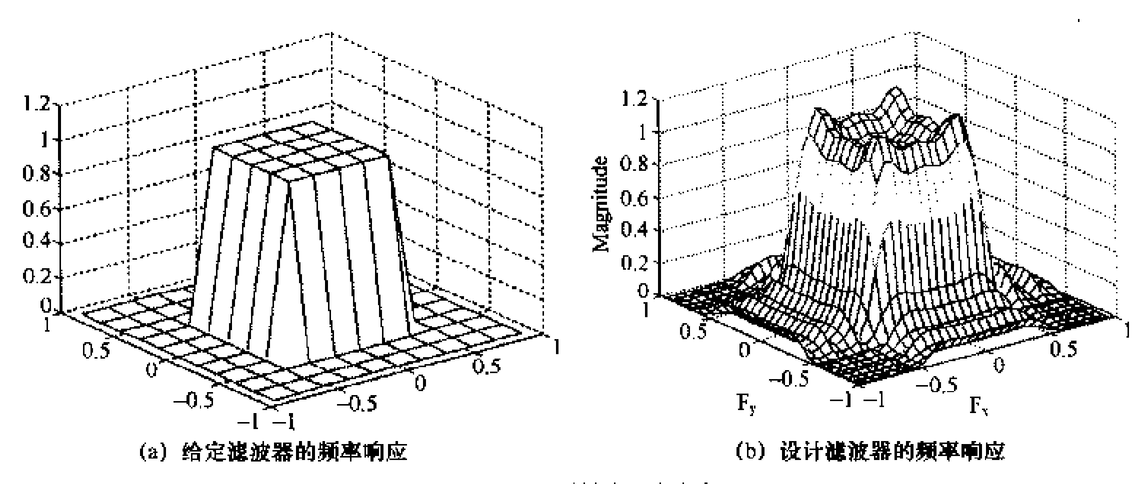
通过以上程序可知，利用函数 ftrans2 可以使用信号处理和滤波器设计工具箱中所有的设计函数进行各种 2-D 图像滤波器设计。例如，先调用函数 butter 进行 1-D 低通、高通、带通或带阻巴特沃斯滤波器设计，然后将其变换为 2-D 低通、高通、带通或带阻巴特沃斯滤波器，而后用于特定的图像处理操作。

10.2.3　频率采样法

利用频率采样法也可以设计满足频率响应要求的滤波器，该方法是根据给定的频率响应矩阵点，设计一个滤波器，使得频率响应通过这些矩阵点。频率采样法对给定点之间的频率响应并无限制。

工具箱中的 fsamp2 函数可以实现 2-D FIR 滤波器的频率采样设计。函数 fsamp2 返回一个滤波器 h，此滤波器的频率响应通过输入矩阵 Hd 的点。以下程序代码示例用 fsamp2 函数创建了一个 11×11 的滤波器，并绘制了该滤波器的频率响应，如图 10-6 为给定滤波器和设计得到滤波器的频率响应图。

```
Hd = zeros(11,11); Hd(4:8,4:8) = 1;
[f1,f2] = freqspace(11,'meshgrid');
mesh(f1,f2,Hd), axis([-1 1 -1 1 0 1.2]), colormap(jet(64))
h = fsamp2(Hd);
figure, freqz2(h,[32 32]), axis([-1 1 -1 1 0 1.2])
```



(a) 给定滤波器的频率响应　　(b) 设计滤波器的频率响应

图 10-6　频率采样法设计滤波器

通过比较要求的频率响应和实际设计的频率响应可知，在实际生成的滤波器的频率响应中，存在明显的波纹，这些波纹是频率设计法中值得注意的问题，它出现在频率响应图中较明显的过渡带中，对滤波器的频率特性有较大影响。

当然，可以通过使用较大的滤波器来降低波纹的空间区域，但这并不能减小波纹的幅度，

而且需要更多的滤波计算时间。为了得到所需频率响应更加逼近和光滑的近似值，可以考虑使用频率转换法或窗口法。

10.2.4 窗口法

窗口法是将理想的脉冲响应与窗函数相乘来获得滤波器。和频率采样方法类似，窗口法将产生一个频率响应近似于所需频率响应的滤波器，但是窗口法得到的结果比频率采样方法要好。

MATLAB7.0 图像处理工具箱提供了两个函数用于实现基于窗口的滤波器设计：fwind1 和 fwind2。fwind1 函数根据由输入参数指定的一个或两个 1-D 窗口创建一个 2-D 窗口，然后进行 2-D 滤波器设计，fwind2 函数则直接使用指定的 2-D 窗口设计 2-D 滤波器。

fwind1 函数支持两种不同的 2-D 窗口创建方法：一种方法是对一个单独的 1-D 窗口进行变换，使用类似旋转的方法创建一个近似中心对称的 2-D 窗口；另一种方法是对两个 1-D 窗口取外积（outer production）得到一个矩形 2-D 窗口。该函数的调用格式如下：

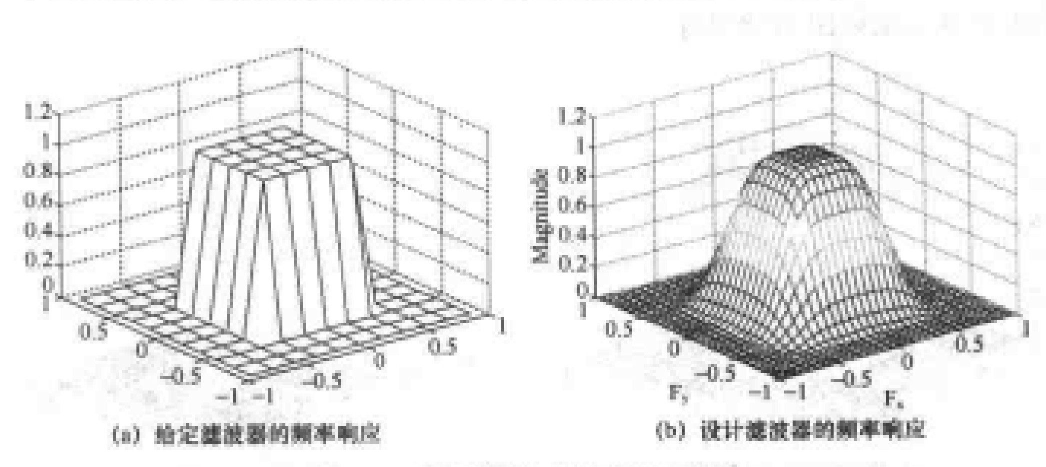
```
h = fwind1(Hd,win)
h = fwind1(Hd,win1,win2)
h = fwind1(f1,f2,Hd,…)
```

h = fwind1(Hd,win)对应着上述窗口创建的第一种方法，Hd 表示指定的频率响应特性，win 是指定的 1-D 窗口函数，可以是任意一种信号处理工具箱中的窗函数，如 boxcar、hamming、hanning、bartlett、blackman、kaiser 或 chebwin 等。返回的滤波器 h 是一个行数和列数都等于 win 长度的方阵。

h = fwind1(Hd,win1,win2)对应着窗口创建的第二种办法，win1 和 win2 分别是两个指定的 1-D 窗口函数，如果 win1 的长度为 n，win2 的长度为 m，则 h 的大小为 n×m。

h = fwind1(f1,f2,Hd,…)中 f1 和 f2 是两个频率矢量，分别用于表示沿 x 轴和 y 轴的指定采样频率点，其元素取值范围是[-1,1]，1 对应于半采样频率点。滤波器 h 的大小与前述两种表达式相同。

例如，以下代码将根据所需的频率响应 Hd 创建一个 11×11 大小的滤波器，其中 hamming 是用于创建一个 1-D“hamming 窗口”的信号处理工具箱函数，fwind1 函数将这个 1-D 窗口拓展为一个 2-D 窗口，所得滤波器的 2-D 频率响应特性如图 10-7 所示。



(a) 给定滤波器的频率响应　　(b) 设计滤波器的频率响应

图 10-7 利用函数 fwind1 生成 2-D 滤波器

```
Hd = zeros(11,11); Hd(4:8,4:8) = 1;
[f1,f2] = freqspace(11,'meshgrid');
mesh(f1,f2,Hd);
axis([-1 1 -1 1 0 1.2]), colormap(jet(64))
h = fwind1(Hd,hamming(11));
figure, freqz2(h,[32 32]);
axis([-1 1 -1 1 0 1.2])
```

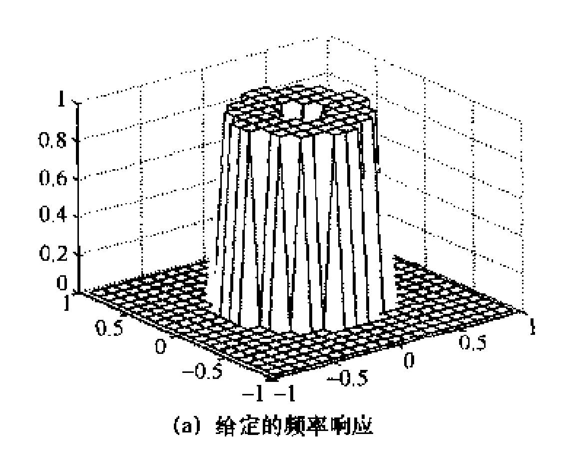
fwind2 函数是对给定的频率响应进行傅立叶反变换，并把结果与指定的 2-D 窗口相乘，从而得到一个 2-D FIR 滤波器。其调用格式如下：

```
h = fwind2(Hd,win)
h = fwind2(f1,f2,Hd,win)
```

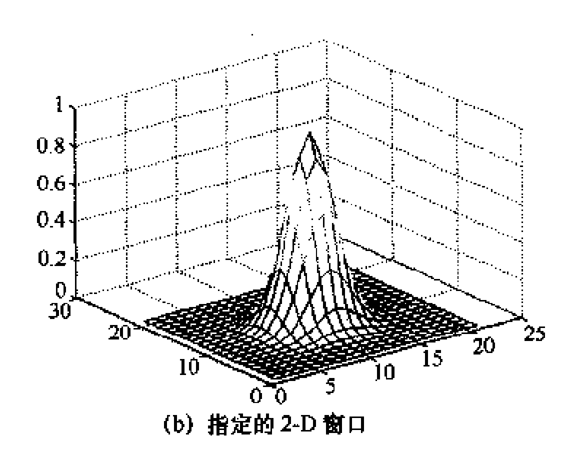
其中，Hd 是一个矩阵，表示给定的（理想的）频率响应，win 是一个 2-D 窗口，h 与 win 大小相同，表示生成的滤波器在数据域空间指定采样点的值。参数 f1 与 f2 的含义与在 fwind1 函数中的对应参数相同。下面给出一个例子，利用函数 fwind2 生成一个 2-D FIR 带通（规定频带：0.1～0.5）滤波器。该例子分三步，实现代码如下：

```
%生成给定的频率响应矩阵 Hd
[f1,f2] = freqspace(21,'meshgrid');
Hd = ones(21);
r = sqrt(f1.^2 + f2.^2);
Hd((r<0.1)|(r>0.5)) = 0;
colormap(jet(64))
mesh(f1,f2,Hd)
%生成指定的窗口
win = fspecial('gaussian',21,2);
win = win ./ max(win(:));    % Make the maximum window value be 1.
mesh(win)
%得到所需的 2-D FIR 滤波器频率
h = fwind2(Hd,win);
freqz2(h)%计算 h 的频率响应
```

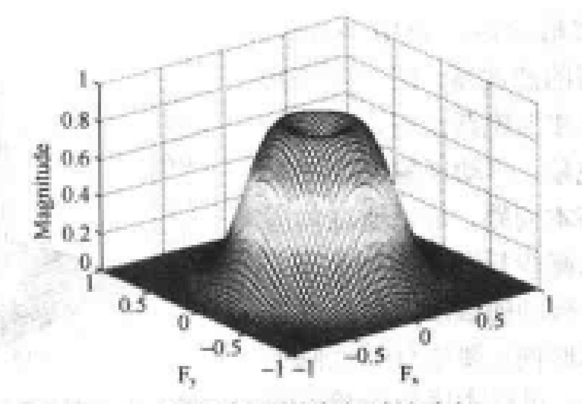
得到的处理结果如图 10-8 所示。



(a) 给定的频率响应



(b) 指定的 2-D 窗口



(c) 设计得到的滤波器频率响应

图 10-8 利用函数 fwind2 设计 2-D FIR 滤波器

由以上两个例子可以看出，相对于频率采样法，窗口法对滤波器创建过程中的波纹有了非常好的抑制作用。

10.2.5 创建满足频率响应要求的矩阵

滤波器设计函数 fsamp2、fwind1 和 fwind2 都是基于所需频率响应幅度矩阵来设计滤波器的，为此，在使用上述函数生成具有特定频率响应的滤波器之前，必须首先生成一个理想的滤波频率响应，这一工作可以通过使用 freqspace 函数来完成。freqspace 函数可以生成任意大小的频率响应矩阵，其元素值对应着正确的和等间距的频率值。当然，我们也可以通过使用频率点来生成理想的滤波频率响应，而不是使用 freqspace 函数的返回值，但是，这种使用频率点的方法不一定能得到满意的结果，如非线性相位。

函数 freqspace 的调用格式为

```
[f1,f2] = freqspace(n)
[f1,f2] = freqspace([m n])
[x1,y1] = freqspace(…,'meshgrid')
```

[f1,f2] = freqspace(n)表示返回长度为 n 的等距离频率点的频率矢量 f1 和 f2。[f1,f2] = freqspace([m n])则指定返回的频率矢量 f1 长度为 n，f2 的长度为 m。当 n 为奇数时，f1 为[-n+1:2:n-1]/n；当 n 为偶数时，f1 为[-n:2:n-2]/n。对于 m 和 f2 有相似的设定。

[x1,y1] = freqspace(…,'meshgrid')等效于

```
[f1,f2] = freqspace(…);
[x1,y1] = meshgrid(f1,f2);
```

返回两个矩阵 x1 和 y1。以[x1,y1] = freqspace([m n],'meshgrid')为例，x1 和 y1 都为 m×n 的矩阵，而且，x1 的列矢量的元素都相等，y1 的行矢量的元素都相等。

以下示例完成设计一个截止频率为 0.5 的理想低通滤波器。生成的滤波器频率响应特性如图 10-9 所示。

```
[f1,f2] = freqspace(25,'meshgrid');
Hd = zeros(25,25); d = sqrt(f1.^2 + f2.^2) < 0.5;
Hd(d) = 1;
mesh(f1,f2,Hd)
```

将所需的频率响应离散化后，调用 fsamp2 函数就可以实现任意所需的滤波器。应当注意的是，当所需频率响应中存在尖锐跃迁时，真实的频率响应就会出现抖动现象。抖动现象是频率采样设计法所需要解决的基本问题，可以通过使用一个带宽较大的滤波器来减少抖动的空间范围。但是，大滤波器不会减小抖动的强度，而且进行滤波时将需要更多的计算时间。要想获得近似于所需频率响应的光滑结果，可以考虑采用频率变换法和窗口法。

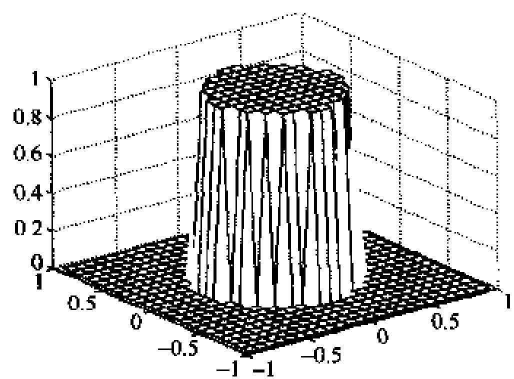


图 10-9　利用函数 freqspace 得到的理想低通滤波器（截止频率为 0.5）

附录 MATLAB 图像处理工具箱函数

表 1　　通 用 函 数

函　　数	功　　能	语　　法
colorbar	显示颜色条	colorbar colorbar(…,'peer',axes_handle) colorbar(axes_handle) colorbar('location') colorbar(…,'PropertyName',propertyvalue) cbar_axes = colorbar(…)
getimage	从坐标轴取得图像数据	A = getimage(h) [x,y,A] = getimage(h) […,A,flag] = getimage(h) […] = getimage
image	创建并显示图像对象	image(C) image(x,y,C) image(…,'PropertyName',PropertyValue,…) image('PropertyName',PropertyValue,…) Formal syntax - PN/PV only handle = image(…)
imagesc	按图像显示数据矩阵	imagesc(C) imagesc(x,y,C) imagesc(…,clims) h = imagesc(…)
imshow	显示图像	imshow(I,n) imshow(I,[low high]) imshow(BW) imshow(X,map) imshow(RGB) imshow(…,display_option) imshow(x,y,A,…) imshow filename h = imshow(…)
imview	利用图像浏览器显示图像	imview(I) imview(RGB) imview(X,map) imview(I,range) imview(filename) imview(…,'InitialMagnification',initial_mag) h = imview(…) imview close all
montage	在矩形框中同时显示多帧图像	montage(I) montage(BW) montage(X,map) montage(RGB) h = montage(…)

续表

函　数	功　能	语　法
immovie	创建多帧索引色图像的电影动画	mov = immovie(X,map) mov = immovie(RGB)
subimage	在一个图形中显示多个图像，结合函数 subplot 使用	subimage(X,map) subimage(I) subimage(BW) subimage(RGB) subimage(x,y,…) h = subimage(…)
truesize	调整图像显示尺寸	truesize(fig,[mrows mcols]) truesize(fig)
wrap	将图像显示到纹理映射表面	warp(X,map) warp(I,n) warp(BW) warp(RGB) warp(z,…) warp(x,y,z,…) h = warp(…)
zoom	缩放图像或图形	zoom on zoom off zoom out zoom reset zoom zoom xon zoom yon zoom(factor) zoom(fig, option)

表 2　　图像文件 I/O 函数

函　数	功　能	语　法
imfinfo	返回图像文件信息	info = imfinfo(filename,fmt) info = imfinfo(filename)
imread	从图像文件中读取图像	A = imread(filename,fmt) [X,map] = imread(filename,fmt) […] = imread(filename) […] = imread(URL,…) […] = imread(…,idx) (CUR, GIF, ICO, and TIFF only) […] = imread(…,'PixelRegion', { ROWS, COLS }) (TIFF only) […] = imread(…,'frames',idx) (GIF only) […] = imread(…,ref)　(HDF only) […] = imread(…,'BackgroundColor',BG) (PNG only) [A,map,alpha] = imread(…) (ICO, CUR, and PNG only)
imwrite	把图像写入图像文件中	imwrite(A,filename,fmt) imwrite(X,map,filename,fmt) imwrite(…,filename) imwrite(…,Param1,Val1,Param2,Val2…)

表 3　　空间变换函数

函　数	功　能	语　法
findbounds	为空间变换寻找输出边界	outbounds = findbounds(TFORM,inbounds)
fliptform	切换空间变换结构的输入和输出角色	TFLIP = fliptform(T)
imcrop	剪切图像	I2 = imcrop(I) X2 = imcrop(X,map) RGB2 = imcrop(RGB) I2 = imcrop(I,rect) X2 = imcrop(X,map,rect) RGB2 = imcrop(RGB,rect) […] = imcrop(x,y,…) [A,rect] = imcrop(…) [x,y,A,rect] = imcrop(…)
imresize	图像缩放	B = imresize(A,m) B = imresize(A,m,method) B = imresize(A,[mrows ncols],method) B = imresize(…,method,n) B = imresize(…,method,h)
imrotate	图像旋转	B = imrotate(A,angle) B = imrotate(A,angle,method) B = imrotate(A,angle,method,bbox)
interp2	2-D 数据插值	ZI = interp2(X,Y,Z,XI,YI) ZI = interp2(Z,XI,YI) ZI = interp2(Z,ntimes) ZI = interp2(X,Y,Z,XI,YI,method)
imtransform	对图像进行 2-D 空间变换	B = imtransform(A,TFORM) B = imtransform(A,TFORM,INTERP) [B,XDATA,YDATA] = imtransform(…) [B,XDATA,YDATA] = imtransform(…, param1, val1, param2, val2,…)
makeresampler	生成重采样结构	R = makeresampler(interpolant,padmethod)
maketform	生成几何变换结构	T = maketform(transformtype,…)
tformarray	多维数组的空间变换	B = tformarray(A, T, R, TDIMS_A, TDIMS_B, TSIZE_B, TMAP_B,F)
tformfwd	正向空间变换	[X,Y] = tformfwd(T,U,V) [X1,X2,X3,…] = tformfwd(T,U1,U2,U3,…) X = tformfwd(T,U) [X1,X2,X3,…] = tformfwd(T,U) X = tformfwd(T,U1,U2,U3,…)
tforminv	逆向空间变换	U = tforminv(X,T)

表 4　　像素和统计处理函数

函　数	功　能	语　法
corr2	计算两个矩阵的 2-D 相关系数	r = corr2(A,B)
imcontour	创建图像的轮廓图	imcontour(I) imcontour(I,n) imcontour(I,v) imcontour(x,y,…) imcontour(…,LineSpec) [C,h] = imcontour(…)

续表

函　　数	功　　能	语　　法
imhist	显示图像的直方图	imhist(I,n) imhist(X,map) [counts,x] = imhist(…)
impixel	确定像素颜色值	P = impixel(I) P = impixel(X,map) P = impixel(RGB) P = impixel(I,c,r) P = impixel(X,map,c,r) P = impixel(RGB,c,r) [c,r,P] = impixel(…) P = impixel(x,y,I,xi,yi) P = impixel(x,y,X,map,xi,yi) P = impixel(x,y,RGB,xi,yi) [xi,yi,P] = impixel(x,y,…)
improfile	沿线段计算剖面图的像素值	c = improfile c = improfile(n) c = improfile(I,xi,yi) c = improfile(I,xi,yi,n) [cx,cy,c] = improfile(…) [cx,cy,c,xi,yi] = improfile(…) […] = improfile(x,y,I,xi,yi) […] = improfile(x,y,I,xi,yi,n) […] = improfile(…,method)
mean2	求矩阵元素平均值	B = mean2(A)
pixval	显示图像像素信息	pixval on pixval off pixval pixval(fig,option) pixval(ax,option) pixval(H,option)
regionprops	得到图像区域属性	STATS = regionprops(L,properties)
std2	计算矩阵元素的标准偏移	b = std2(A)

表 5　　图像分析函数

函　　数	功　　能	语　　法
edge	识别灰度图像中的边界	BW = edge(I,'sobel') BW = edge(I,'sobel',thresh) BW = edge(I,'sobel',thresh,direction) [BW,thresh] = edge(I,'sobel',…) BW = edge(I,'prewitt') BW = edge(I,'prewitt',thresh) BW = edge(I,'prewitt',thresh,direction) [BW,thresh] = edge(I,'prewitt',…) BW = edge(I,'roberts') BW – edge(I,'roberts',thresh) [BW,thresh] = edge(I,'roberts',…) BW = edge(I,'log') BW = edge(I,'log',thresh) BW = edge(I,'log',thresh,sigma) [BW,threshold] = edge(I,'log',…)

续表

函　　数	功　　能	语　　法
qtdecomp	执行四叉树分解	S = qtdecomp(I) S = qtdecomp(I,threshold) S = qtdecomp(I,threshold,mindim) S = qtdecomp(I,threshold,[mindim maxdim]) S = qtdecomp(I,fun) S = qtdecomp(I,fun,P1,P2,…)
qtgetblk	获取四叉树分解中的数组块值	[vals,r,c] = qtgetblk(I,S,dim) [vals,idx] = qtgetblk(I,S,dim)
qtsetblk	设置四叉树分解中的数组块值	J = qtsetblk(I,S,dim,vals)

表 6　　图像增强函数

函　　数	功　　能	语　　法
adapthisteq	执行对比度受限的直方图均衡	J = adapthisteq(I) J = adapthisteq(I,param1,val1,param2,val2…)
decorrstretch	对多通道图像应用解卷积延拓	S = decorrstretch(I) S = decorrstretch(I,TOL)
histeq	用直方图均等化增强对比度	J = histeq(I,hgram) J = histeq(I,n) [J,T] = histeq(I,…) newmap = histeq(X,map,hgram) newmap = histeq(X,map) [newmap,T] = histeq(X,…)
imadjust	调整图像灰度值或颜色映射表	J = imadjust(I) J = imadjust(I,[low_in; high_in],[low_out; high_out]) J = imadjust(…,gamma) newmap = imadjust(map, [low_in high_in], [low_out high_out], gamma) RGB2 = imadjust(RGB1,…)
imnoise	向图像中加入噪声	J = imnoise(I,type) J = imnoise(I,type,parameters)
medfilt2	进行二维中值滤波	B = medfilt2(A,[m n]) B = medfilt2(A) B = medfilt2(A,'indexed',…)
ordfilt2	进行二维统计顺序滤波	B = ordfilt2(A,order,domain) B = ordfilt2(A,order,domain,S) B = ordfilt2(…,padopt)
stretchlim	得到图像对比度延拓的灰度上下限	LOW_HIGH = stretchlim(I,TOL) LOW_HIGH = stretchlim(RGB,TOL)
wiener2	进行二维适应性去噪滤波	J = wiener2(I,[m n],noise) [J,noise] = wiener2(I,[m n])

表 7　　线性滤波函数

函　　数	功　　能	语　　法
conv2	二维卷积	C = conv2(A,B) C = conv2(hcol,hrow,A) C = conv2(…,'shape')
convmtx2	二维矩阵卷积	T = convmtx2(H,m,n) T = convmtx2(H,[m n])

续表

函　数	功　能	语　法
convn	n 维卷积	C = convn(A,B) C = convn(A,B,'shape')
filter2	二维线性滤波	Y = filter2(h,X) Y = filter2(h,X,shape)
fspecial	创建预定义滤波器	h = fspecial(type) h = fspecial(type,parameters)
imfilter	多维图像滤波	B = imfilter(A,H) B = imfilter(A,H,option1,option2,⋯)

表 8　　线性二维滤波器设计函数

函　数	功　能	语　法
freqspace	确定二维频率响应的频率空间	[f1,f2] = freqspace(n) [f1,f2] = freqspace([m n]) [x1,y1] = freqspace(⋯,'meshgrid') f = freqspace(N) f = freqspace(N,'whole')
freqz2	计算二维频率响应	[H,f1,f2] = freqz2(h,n1,n2) [H,f1,f2] = freqz2(h,[n2 n1]) [H,f1,f2] = freqz2(h) [H,f1,f2] = freqz2(h,f1,f2) [⋯] = freqz2(h,⋯,[dx dy]) [⋯] = freqz2(h,⋯,dx) freqz2(⋯)
fsamp2	用频率采样法设计二维 FIR 滤波器	h = fsamp2(Hd) h = fsamp2(f1,f2,Hd,[m n])
ftrans2	通过频率转换法设计二维 FIR 滤波器	h = ftrans2(b,t) h = ftrans2(b)
fwind1	用一维窗口方法设计二维 FIR 滤波器	h = fwind1(Hd,win) h = fwind1(Hd,win1,win2) h = fwind1(f1,f2,Hd,⋯)
fwind2	用二维窗口方法设计二维 FIR 滤波器	h = fwind2(Hd,win) h = fwind2(f1,f2,Hd,win)

表 9　　图像变换函数

函　数	功　能	语　法
dct2	进行二维离散余弦变换	B = dct2(A) B = dct2(A,m,n) B = dct2(A,[m n])
dctmtx	计算离散余弦变换矩阵	D = dctmtx(n)
fft2	进行二维快速傅立叶变换	Y = fft2(X) Y = fft2(X,m,n)
fftn	进行 n 维快速傅立叶变换	Y = fftn(X) Y = fftn(X,siz)
fftshift	转换快速傅立叶变换的输出象限	Y = fftshift(X) Y = fftshift(X,dim)

续表

函　　数	功　　能	语　　法
idct2	计算二维逆离散余弦变换	B = idct2(A) B = idct2(A,m,n) B = idct2(A,[m n])
ifft2	计算二维逆快速傅立叶变换	Y = ifft2(X) Y = ifft2(X,m,n) y = ifft2(…, 'nonsymmetric') y = ifft2(…, 'nonsymmetric')
ifftn	计算 n 维逆快速傅立叶变换	Y = ifftn(X) Y = ifftn(X,siz) y = ifftn(…, 'nonsymmetric') y = ifftn(…, 'nonsymmetric')
iradon	逆 Radon 变换	I = iradon(R,theta) I = iradon(R, theta, interp, filter, frequency_scaling, output_size) [I,H] = iradon(…)
phantom	产生一个头部幻影图像	P = phantom(def,n) P = phantom(E,n) [P,E] = phantom(…)
radon	计算 Radon 变换	R=radon(I,theta) [R,xp]=radon(…)
fanbeam	计算扇形投影变换	F = fanbeam(I,D) F = fanbeam(…,param1,val1,param1,val2,…) [F,sensor_positions,fan_rotation_angles] = fanbeam(…)

表 10　　　　边沿和块处理函数

函　　数	功　　能	语　　法
bestblk	确定进行块操作的块大小	siz = bestblk([m n],k) [mb,nb] = bestblk([m n],k)
blkproc	实现图像的非重叠（distinct）块操作	B = blkproc(A,[m n],fun) B = blkproc(A,[m n],fun,P1,P2,…) B=blkproc(A,[m n],[mborder nborder], fun,…) B = blkproc(A,'indexed',…)
col2im	将矩阵的列重新组织到块中	A = col2im(B,[m n],[mm nn], block_type) A = col2im(B,[m n],[mm nn])
colfilt	利用列相关函数进行边沿操作	B = colfilt(A,[m n],block_type,fun) B = colfilt(A,[m n],block_type,fun,P1,P2,…) B = colfilt(A, [m n], [mblock nblock], block_type, fun,…) B = colfilt(A,'indexed',…)
im2col	重调图像块为列	B = im2col(A,[m n],block_type) B = im2col(A,[m n]) B = im2col(A,'indexed',…)
nlfilter	通用滑动邻域操作	B = nlfilter(A,[m n],fun) B = nlfilter(A,[m n],fun,P1,P2,…) B = nlfilter(A,'indexed',…)

表 11　　　　　　　　　　　　　　图像形态学操作函数

函　数	功　能	语　法
applylut	在二值图像中利用查找表进行邻域操作	A = applylut(BW,LUT)
bwarea	计算二值图像的对象面积	total = bwarea(BW)
bweuler	计算二值图像的欧拉数	eul = bweuler(BW,n)
bwhitmiss	执行二值图像的击中和击不中操作	BW2 = bwhitmiss(BW1,SE1,SE2) BW2 = bwhitmiss(BW1,INTERVAL)
bwlabel	标注二值图像中已连接的部分	L = bwlabel(BW,n) [L,num] = bwlabel(BW,n)
bwmorph	二值图像的通用形态学操作	BW2 = bwmorph(BW,operation) BW2 = bwmorph(BW,operation,n)
bwperim	计算二值图像中对象的周长	BW2 = bwperim(BW1) BW2 = bwperim(BW1,CONN)
bwselect	在二值图像中选择对象	BW2 = bwselect(BW,c,r,n) BW2 = bwselect(BW,n) [BW2,idx] = bwselect(…) BW2 = bwselect(x,y,BW,xi,yi,n) [x,y,BW2,idx,xi,yi] = bwselect(…)
makelut	创建用于 applylut 函数的查找表	lut = makelut(fun,n) lut = makelut(fun,n,P1,P2,…)
bwdist	距离变换	D = bwdist(BW) [D,L] = bwdist(BW) [D,L] = bwdist(BW,METHOD)
imbothat	执行形态学的闭包运算	IM2 = imbothat(IM,SE) IM2 = imbothat(IM,NHOOD)
imclose	图像的闭运算	IM2 = imclose(IM,SE) IM2 = imclose(IM,NHOOD)
imopen	图像的开运算	IM2 = imopen(IM,SE) IM2 = imopen(IM,NHOOD)
imdilate	图像的膨胀	IM2 = imdilate(IM,SE) IM2 = imdilate(IM,NHOOD) IM2 = imdilate(IM,SE,PACKOPT) IM2 = imdilate(…,PADOPT)
imerode	图像的腐蚀	IM2 = imerode(IM,SE) IM2 = imerode(IM,NHOOD) IM2 = imerode(IM,SE,PACKOPT,M) IM2 = imerode(…,PADOPT)
imfill	填充图像区域	BW2 = imfill(BW,locations) BW2 = imfill(BW,'holes') I2 = imfill(I) BW2 = imfill(BW) [BW2 locations] = imfill(BW) BW2 = imfill(BW,locations,CONN) BW2 = imfill(BW,CONN,'holes') I2 = imfill(I,CONN)
imtophat	用开运算后的图像减去原图像	IM2 = imtophat(IM,SE) IM2 = imtophat(IM,NHOOD)
strel	创建形态学结构元素	SE = strel(shape,parameters)

表 12　　区域处理函数

函　　数	功　　能	语　　法
roicolor	选择感兴趣的颜色区	BW = roicolor(A,low,high) BW = roicolor(A,v)
roifill	在图像的任意区域中进行平滑插补	J = roifill(I,c,r) J = roifill(I) J = roifill(I,BW) [J,BW] = roifill(…) J = roifill(x,y,I,xi,yi) [x,y,J,BW,xi,yi] = roifill(…)
roifilt2	滤波特定区域	J = roifilt2(h,I,BW) J = roifilt2(I,BW,fun) J = roifilt2(I,BW,fun,P1,P2,…)
roipoly	选择一个感兴趣的多边形区域	BW = roipoly(I,c,r) BW = roipoly(I) BW = roipoly(x,y,I,xi,yi) [BW,xi,yi] = roipoly(…) [x,y,BW,xi,yi] = roipoly(…)

表 13　　图像代数操作

函　　数	功　　能	语　　法
imadd	加运算	Z = imadd(X,Y)
imsubtract	减运算	Z = imsubtract(X,Y)
immultiply	乘运算	Z = immultiply(X,Y)
imdivide	除运算	Z = imdivide(X,Y)

表 14　　颜色空间转换函数

函　　数	功　　能	语　　法
hsv2rgb	转换 HSV 的值为 RGB 颜色空间	M = hsv2rgb(H)
ntsc2rgb	转换 NTSC 的值为 RGB 颜色空间	rgbmap = ntsc2rgb(yiqmap) RGB = ntsc2rgb(YIQ)
rgb2hsv	转换 RGB 的值为 HSV 颜色空间	cmap = rgb2hsv(M)
rgb2ntsc	转换 RGB 的值为 NTSC 颜色空间	yiqmap = rgb2ntsc(rgbmap) YIQ = rgb2ntsc(RGB)
rgb2ycbcr	转换 RGB 的值为 YCbCr 颜色空间	ycbcrmap = rgb2ycbcr(rgbmap) YCBCR = rgb2ycbcr(RGB)
ycbcr2rgb	转换 YCbCr 的值为 RGB 颜色空间	rgbmap = ycbcr2rgb(ycbcrmap) RGB = ycbcr2rgb(YCBCR)

表 15　　图像类型和类型转换函数

函　　数	功　　能	语　　法
dither	通过抖动增加外观颜色分辨率，转换图像	X = dither(RGB,map) BW = dither(I)
gray2ind	转换灰度图像为索引色图像	[X,map] = gray2ind(I,n) [X,map] = gray2ind(BW,n)

续表

函　　数	功　　能	语　　法
grayslice	通过设定阈值转化灰度图像为索引色图像	X = grayslice(I,n) X = grayslice(I,v)
im2bw	转换图像为二值图像	BW = im2bw(I,level) BW = im2bw(X,map,level) BW = im2bw(RGB,level)
im2double	转换图像矩阵为双精度类型	I2 = im2double(I) RGB2 = im2double(RGB) I = im2double(BW) X2 = im2double(X,'indexed')
double	转换数据为双精度类型	double(X)
uint8	转换数据为 8 位无符号整型	I = uint8(X)
im2uint8	转换图像阵列为 8 位为无符号整型	I2 = im2uint8(I) RGB2 = im2uint8(RGB) I = im2uint8(BW) X2 = im2uint8(X,'indexed')
im2uint16	转换图像阵列为 16 位为无符号整型	I2 = im2uint16(I) RGB2 = im2uint16(RGB) I = im2uint16(BW) X2 = im2uint16(X,'indexed')
uint16	转换数据为 16 位无符号整型	I = uint16(X)
ind2gray	转换索引色图像为灰度图像	I = ind2gray(X,map)
ind2rgb	转换索引色图像为 RGB 图像	RGB = ind2rgb(X,map)
isbw	判断是否为二值图像	flag = isbw(A)
isgray	判断是否为灰度图像	flag = isgray(A)
isind	判断是否为索引色图像	flag = isind(A)
isrgb	判断是否为 RGB 图像	flag = isrgb(A)
mat2gray	转换矩阵为灰度图像	I = mat2gray(A,[amin amax]) I = mat2gray(A)
rgb2gray	转换 RGB 图像或颜色映射表为灰度图像	I = rgb2gray(RGB) newmap = rgb2gray(map)
rgb2ind	转换 RGB 图像为索引色图像	[X,map] = rgb2ind(RGB,tol) [X,map] = rgb2ind(RGB,n) X = rgb2ind(RGB,map) […] = rgb2ind(…,dither_option)

表 16　　图像复原函数

函　　数	功　　能	语　　法
deconvwnr	用维纳滤波复原图像	J = deconvwnr(I,PSF) J = deconvwnr(I,PSF,NSR) J = deconvwnr(I,PSF,NCORR,ICORR)

续表

函　数	功　能	语　法
deconvreg	用最小约束二乘滤波复原图像	J = deconvreg(I,PSF) J = deconvreg(I,PSF,NOISEPOWER) J = deconvreg(I, PSF, NOISEPOWER, LRANGE) J = deconvreg(I, PSF, NOISEPOWER, LRANGE, REGOP) [J, LAGRA] = deconvreg(I,PSF,⋯)
deconvlucy	用 Richardson-Lucy 滤波复原图像	J = deconvlucy(I,PSF) J = deconvlucy(I,PSF,NUMIT) J = deconvlucy(I,PSF,NUMIT,DAMPAR) J = deconvlucy(I, PSF, NUMIT, DAMPAR, WEIGHT) J = deconvlucy(I, PSF, NUMIT, DAMPAR, WEIGHT, READOUT) J = deconvlucy(I, PSF, NUMIT, DAMPAR, WEIGHT, READOUT, SUBSMPL)
deconvblind	用盲卷积滤波复原图像	[J,PSF] = deconvblind(I,INITPSF) [J,PSF] = deconvblind(I,INITPSF,NUMIT) [J,PSF] = deconvblind(I, INITPSF, NUMIT, DAMPAR) [J,PSF] = deconvblind(I, INITPSF, NUMIT, DAMPAR, WEIGHT) [J,PSF] = deconvblind(I, INITPSF, NUMIT, DAMPAR, WEIGHT, READOUT) [J,PSF] = deconvblind(⋯, FUN, P1, P2,⋯, PN)

参 考 文 献

[1]（美）冈萨雷斯等. 数字图像处理（MATLAB 版）. 阮秋琦等译. 北京：电子工业出版社，2005

[2]（美）冈萨雷斯等. 数字图像处理（第二版）中文版. 阮秋琦等译. 北京：电子工业出版社，2003

[3] 贾永红. 数字图像处理. 武汉：武汉大学出版社，2003

[4] 闫敬文. 数字图像处理技术与图像图形学基本教程. 北京：科学出版社，2002

[5] 阮秋琦. 数字图像处理学. 北京：电子工业出版社，2000

[6] 张志涌. 精通 MATLAB 6.5 版. 北京：北京航空航天大学出版社，2003

[7] 林雪松等. MATLAB 7.0 应用集锦. 北京：机械工业出版社，2006

[8] 苏金明等. MATLAB 实用教程. 北京：电子工业出版社，2005

[9] 肖伟. MATLAB 程序设计与应用. 北京：北京交通大学出版社，2005

[10] 郝文化等. MATLAB 图形图像处理应用教程. 北京：中国水利水电出版社，2004

[11] 孙兆林. MATLAB6.x 图像处理. 北京：清华大学出版社，2002

[12] 罗军辉等. MATLAB7.0 在图像处理中的应用. 北京：机械工业出版社，2005

[13] 飞思科技产品研发中心. MATLAB6.5 辅助图像处理. 北京：电子工业出版社，2003

[14] 孙即祥. 数字图像处理. 石家庄：河北教育出版社，1993

[15] MATLAB7.0 帮助文档